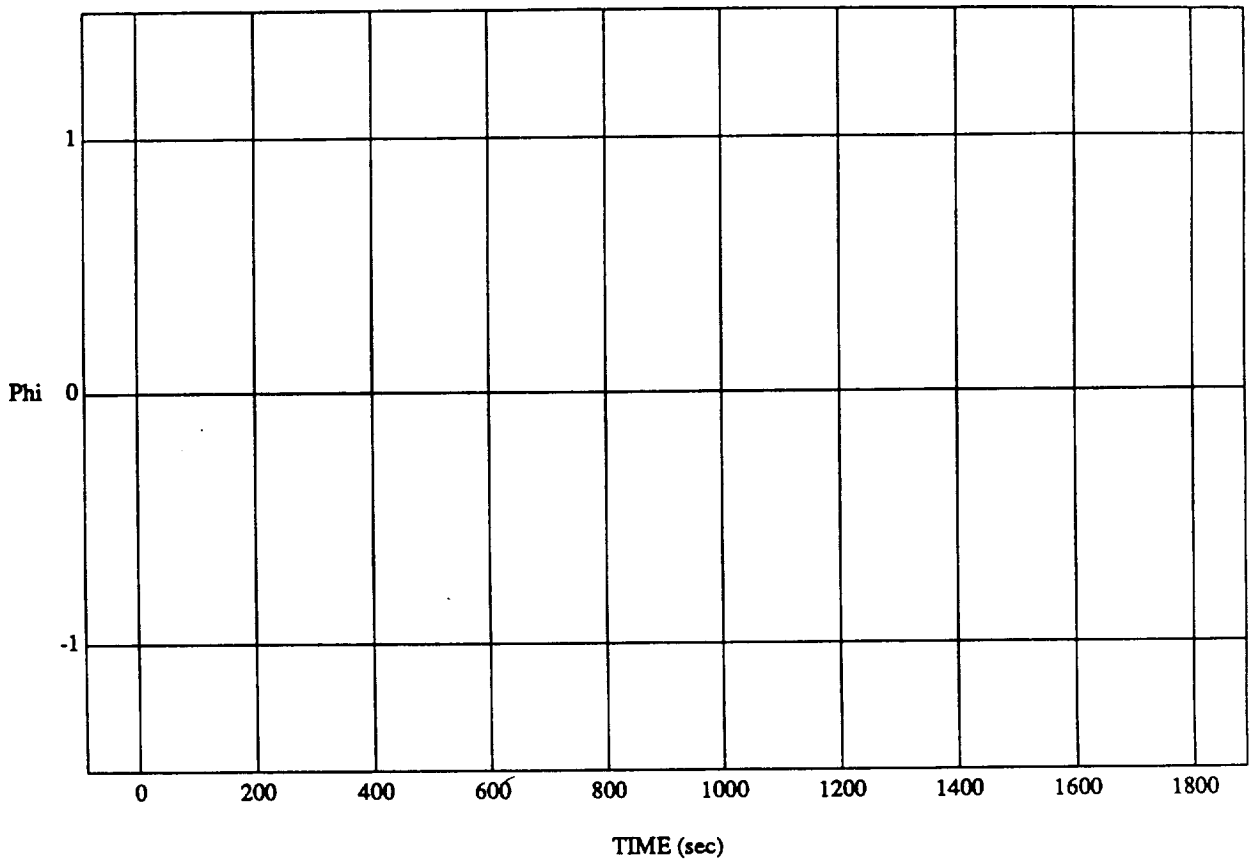
Appendix A.

Final Source Listing of Fuzzy Learning Modules for Shuttle Translational Controller

SIMULATION APPLICATION: ARIC Translational Controller Simulation

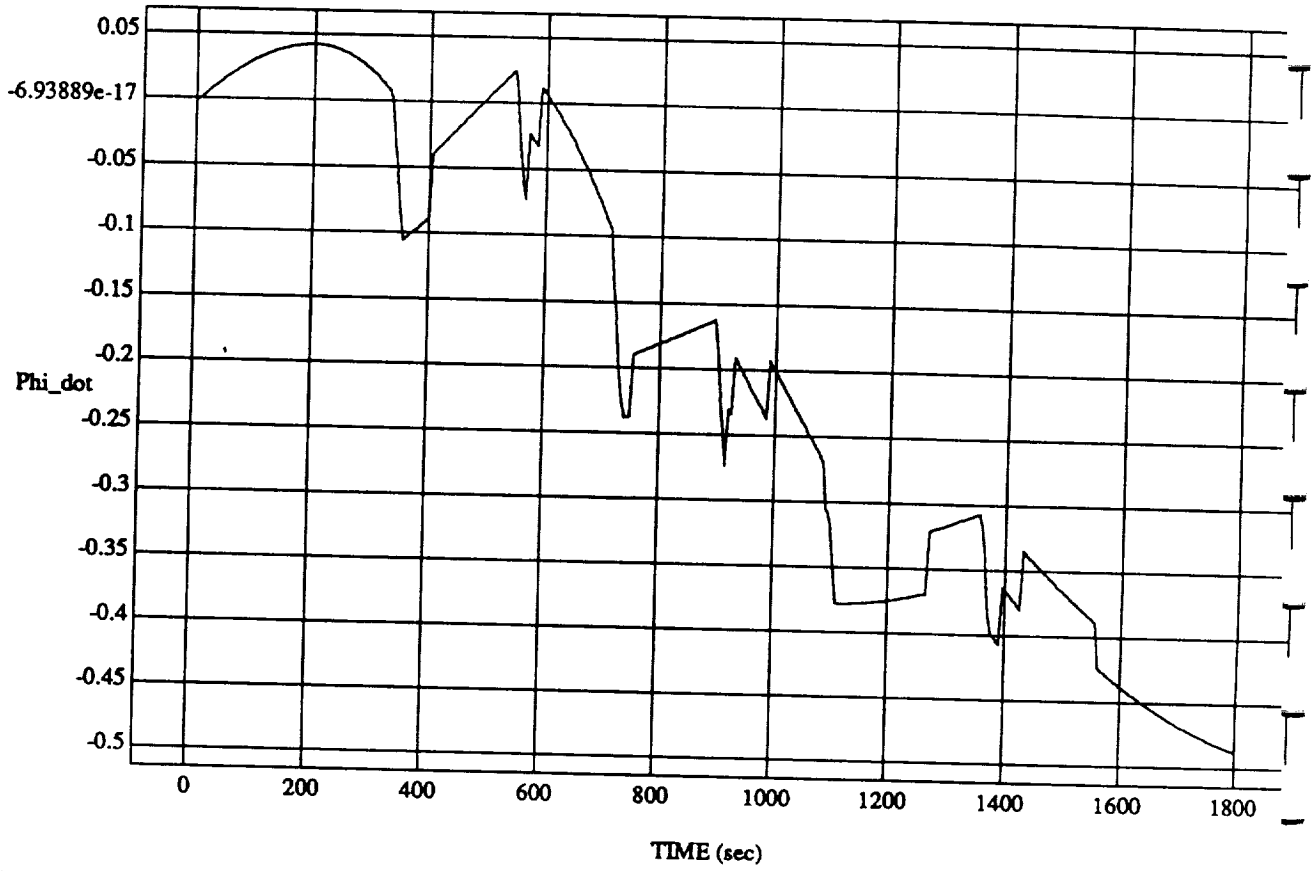
Phi vs TIME

RUN: V Bar Approach



MODULE: ORBITER.lm_range
DATA SAMPLING FREQUENCY: 0.200 Hz

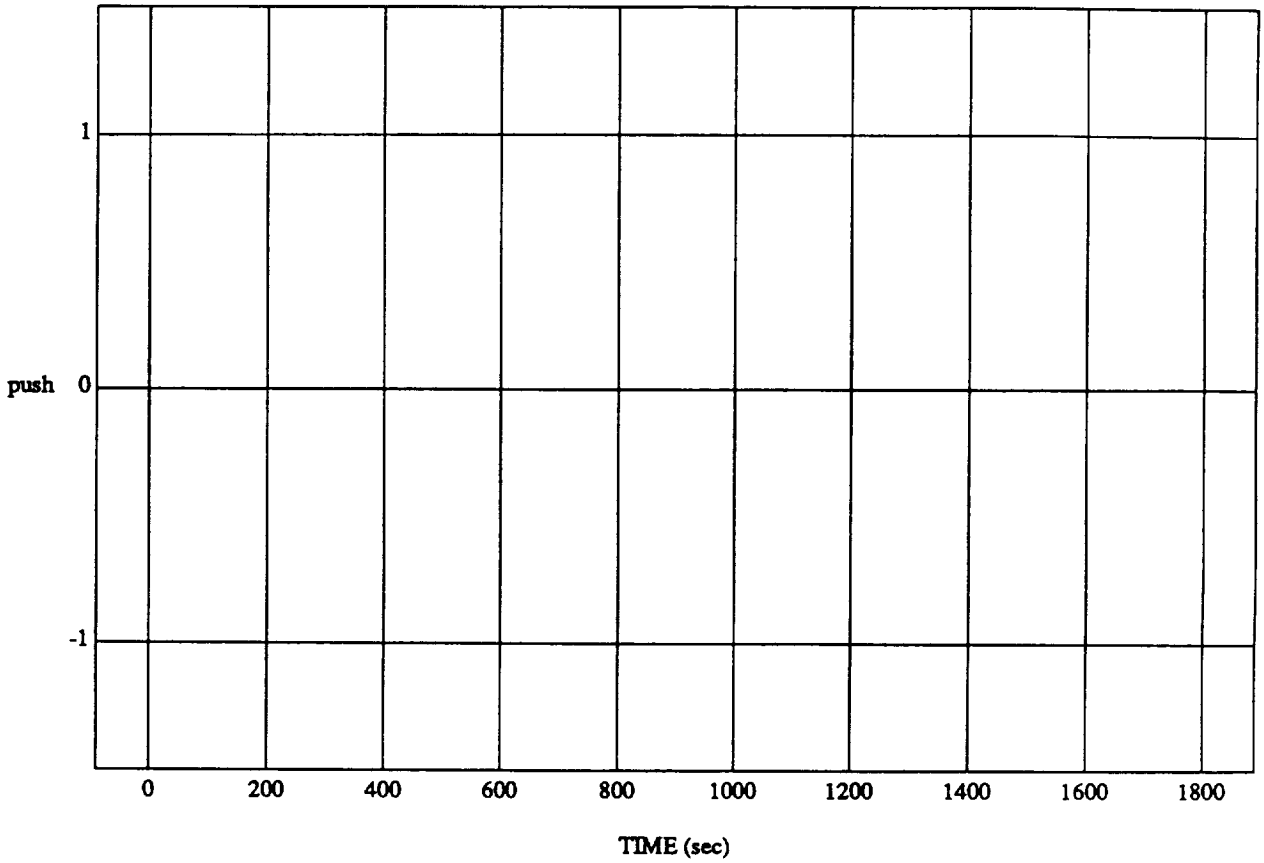
Phi_dot vs TIME
RUN: V Bar Approach



MODULE: ORBITER.lm_range
DATA SAMPLING FREQUENCY: 0.200 Hz

push vs TIME

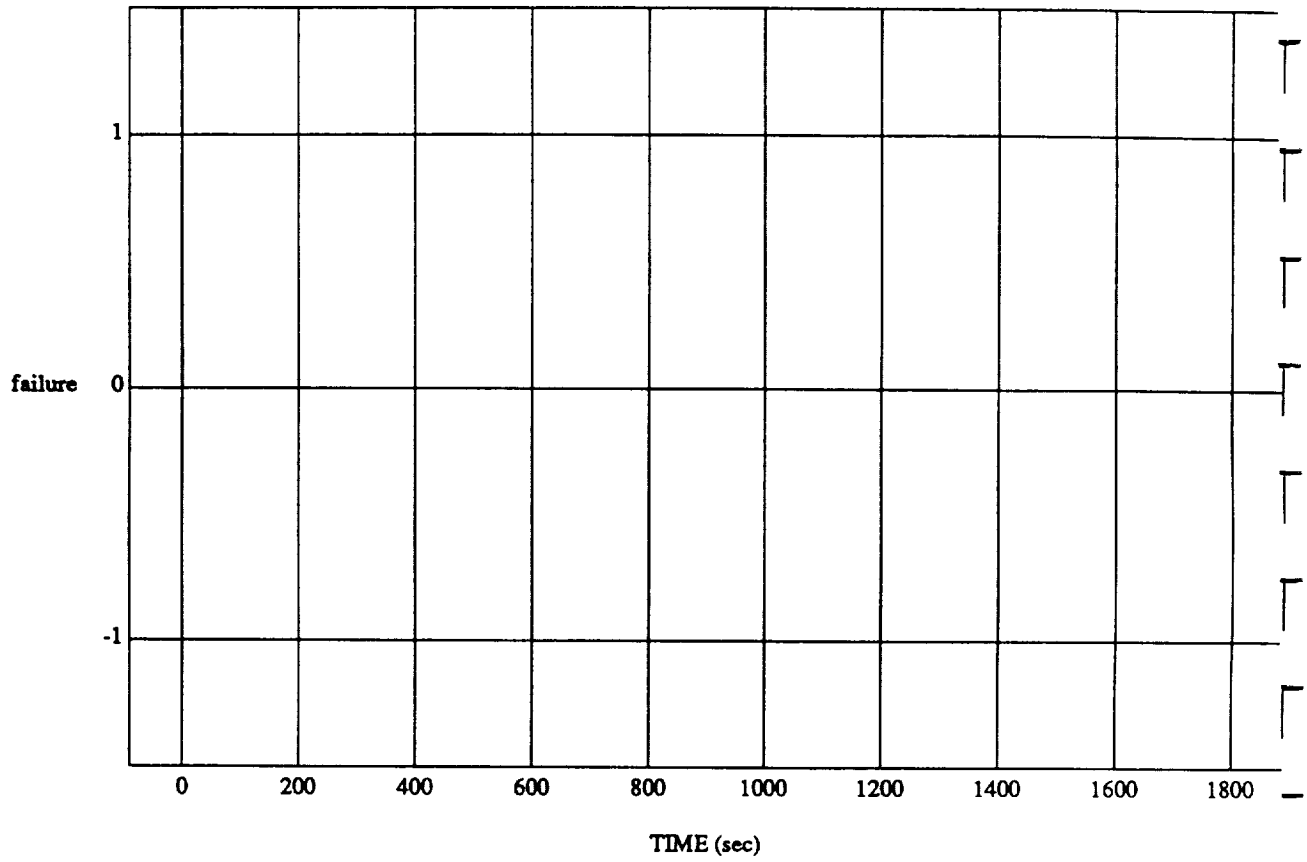
RUN: V Bar Approach



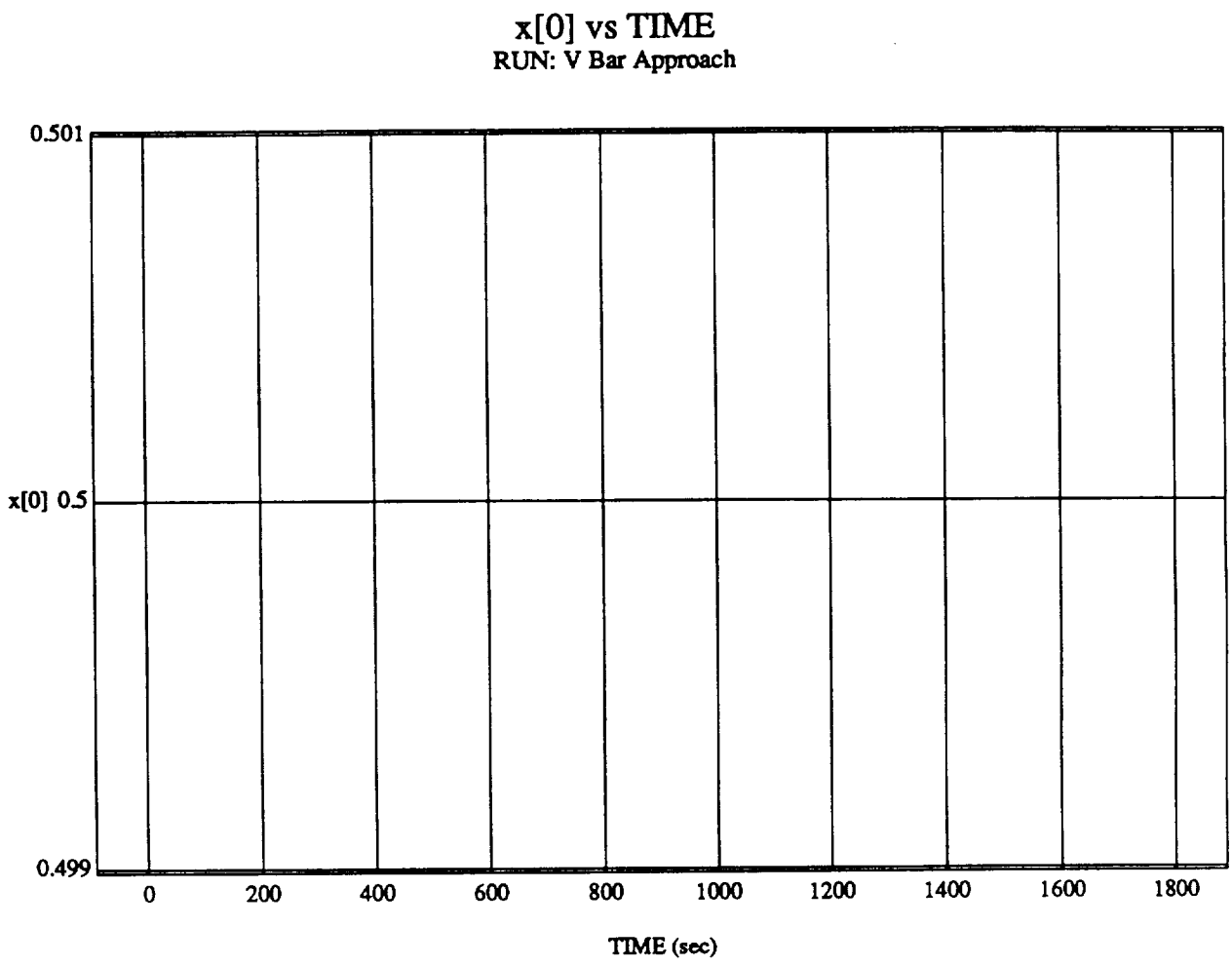
MODULE: ORBITER.lm_range
DATA SAMPLING FREQUENCY: 0.200 Hz

failure vs TIME

RUN: V Bar Approach

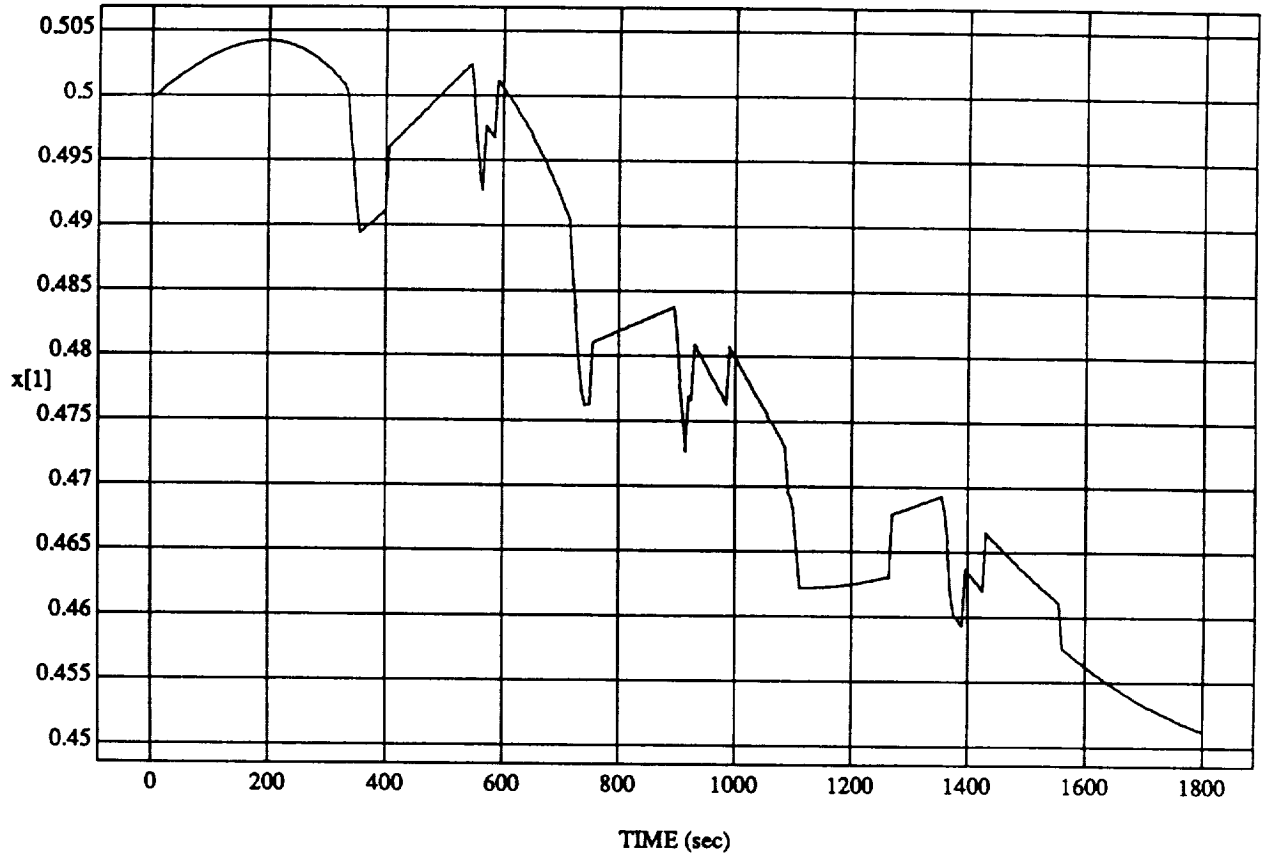


MODULE: ORBITER.lm_range
DATA SAMPLING FREQUENCY: 0.200 Hz



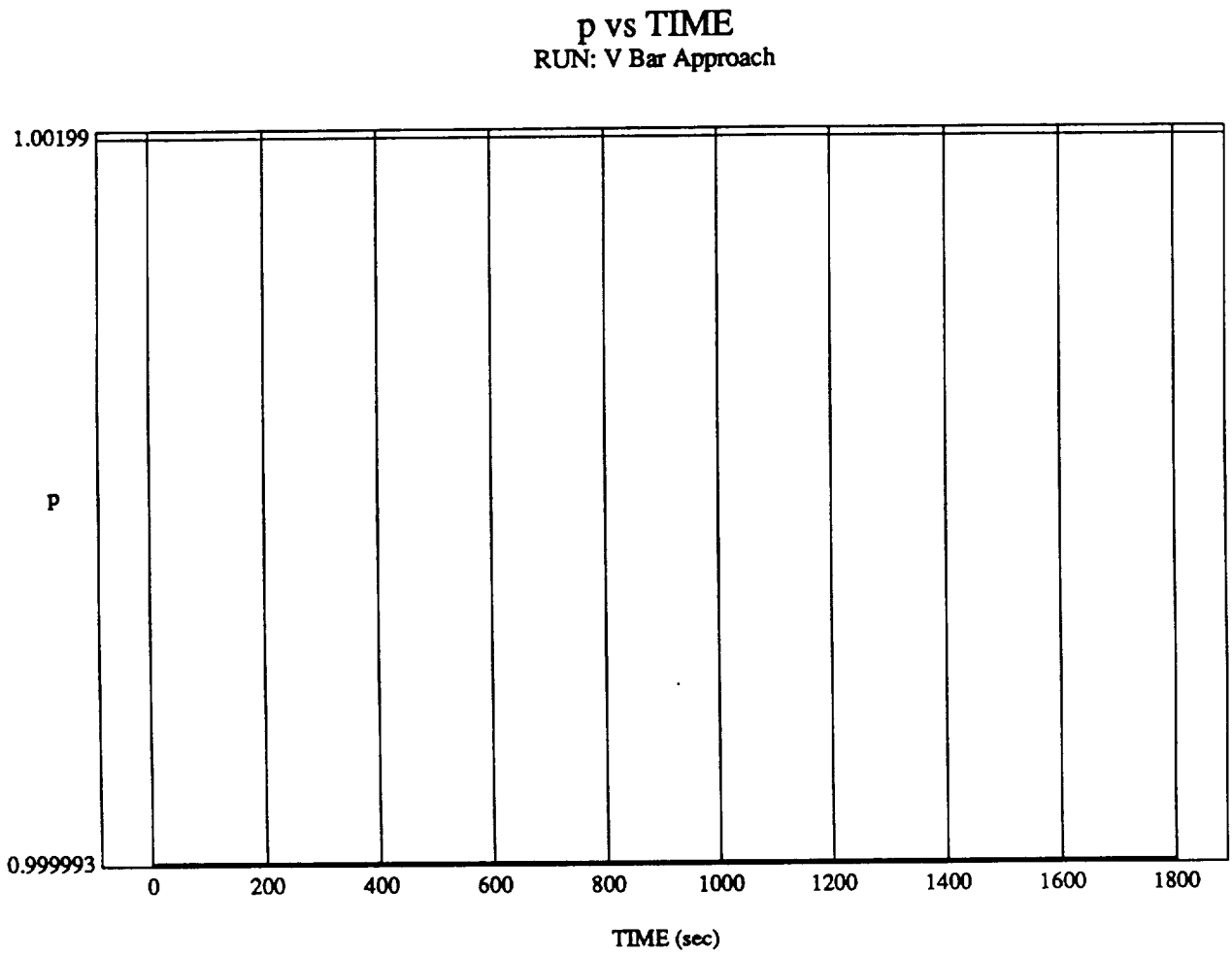
MODULE: ORBITER.lm_range
DATA SAMPLING FREQUENCY: 0.200 Hz

x[1] vs TIME
RUN: V Bar Approach



MODULE: ORBITER.lm_range
DATA SAMPLING FREQUENCY: 0.200 Hz

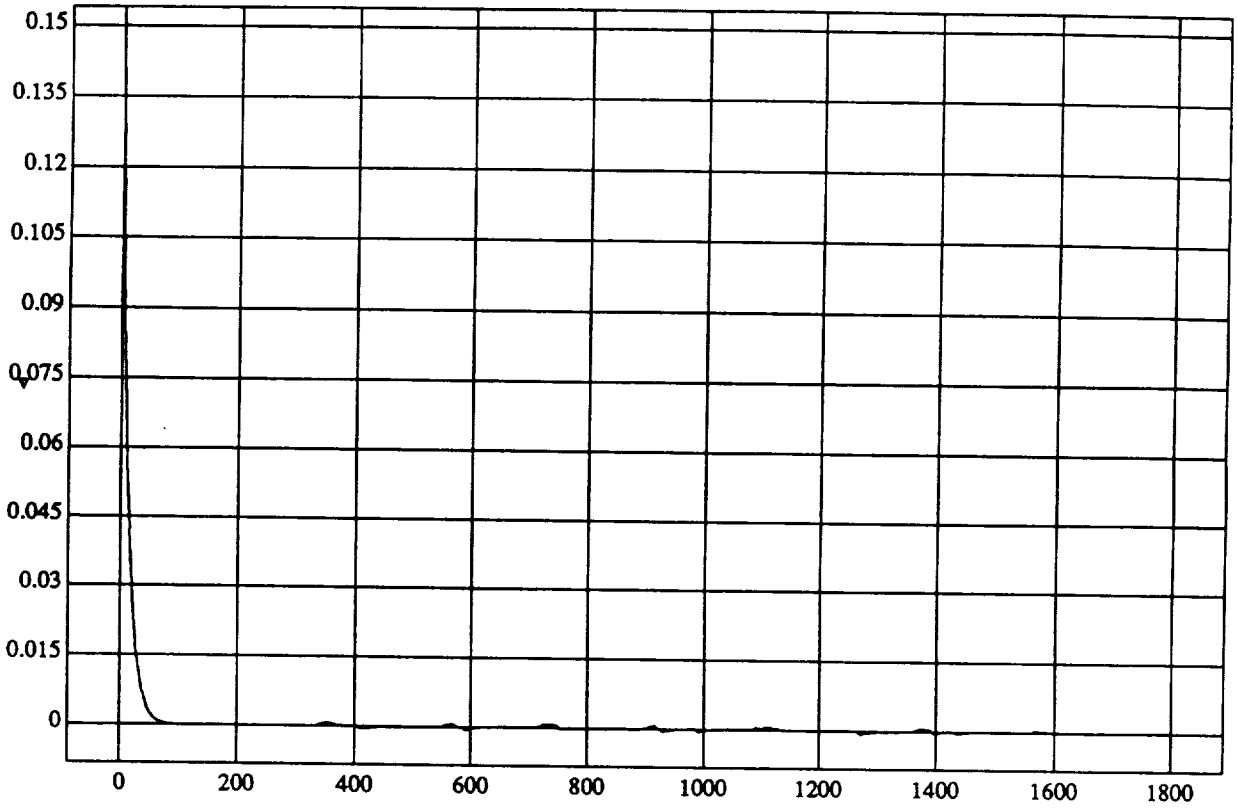
SIMULATION APPLICATION: ARIC Translational Controller Simulation



MODULE: ORBITER.lm_range
DATA SAMPLING FREQUENCY: 0.200 Hz

SIMULATION APPLICATION: ARIC Translational Controller Simulation

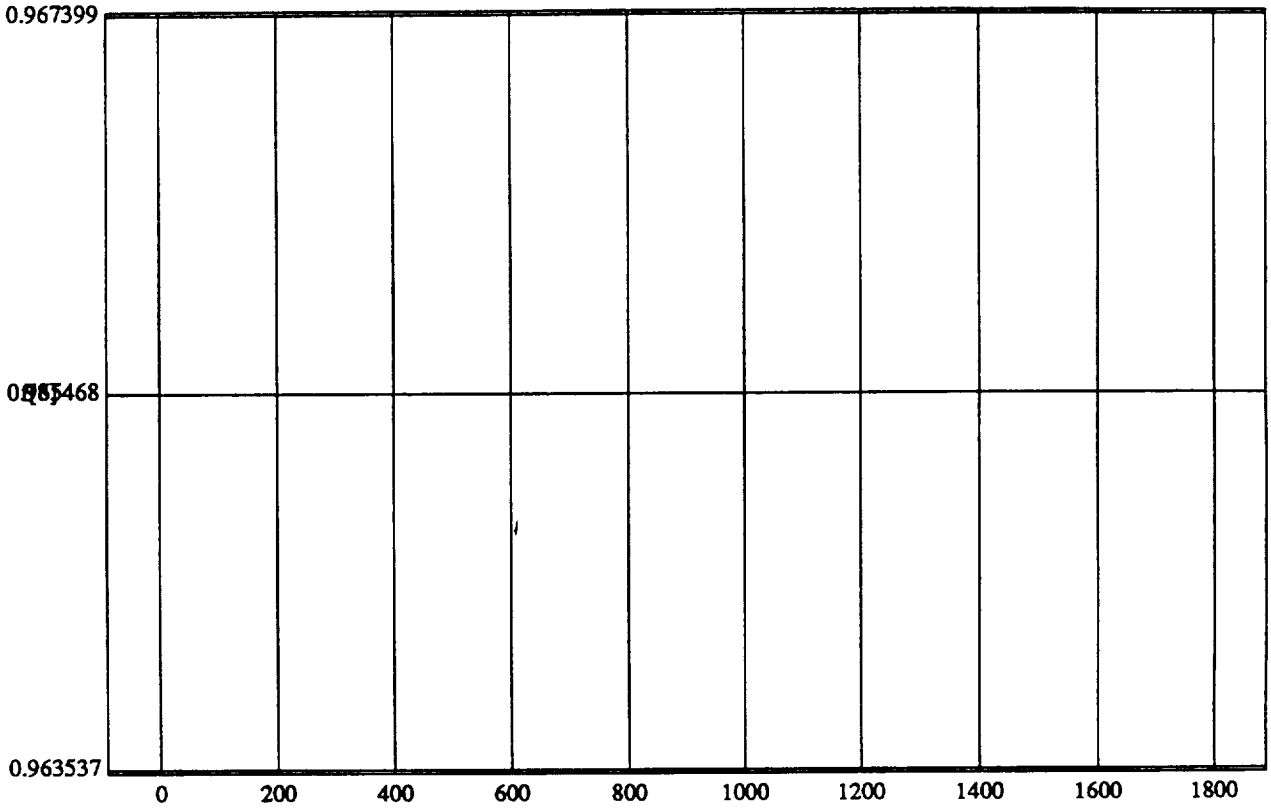
v vs TIME
RUN: V Bar Approach



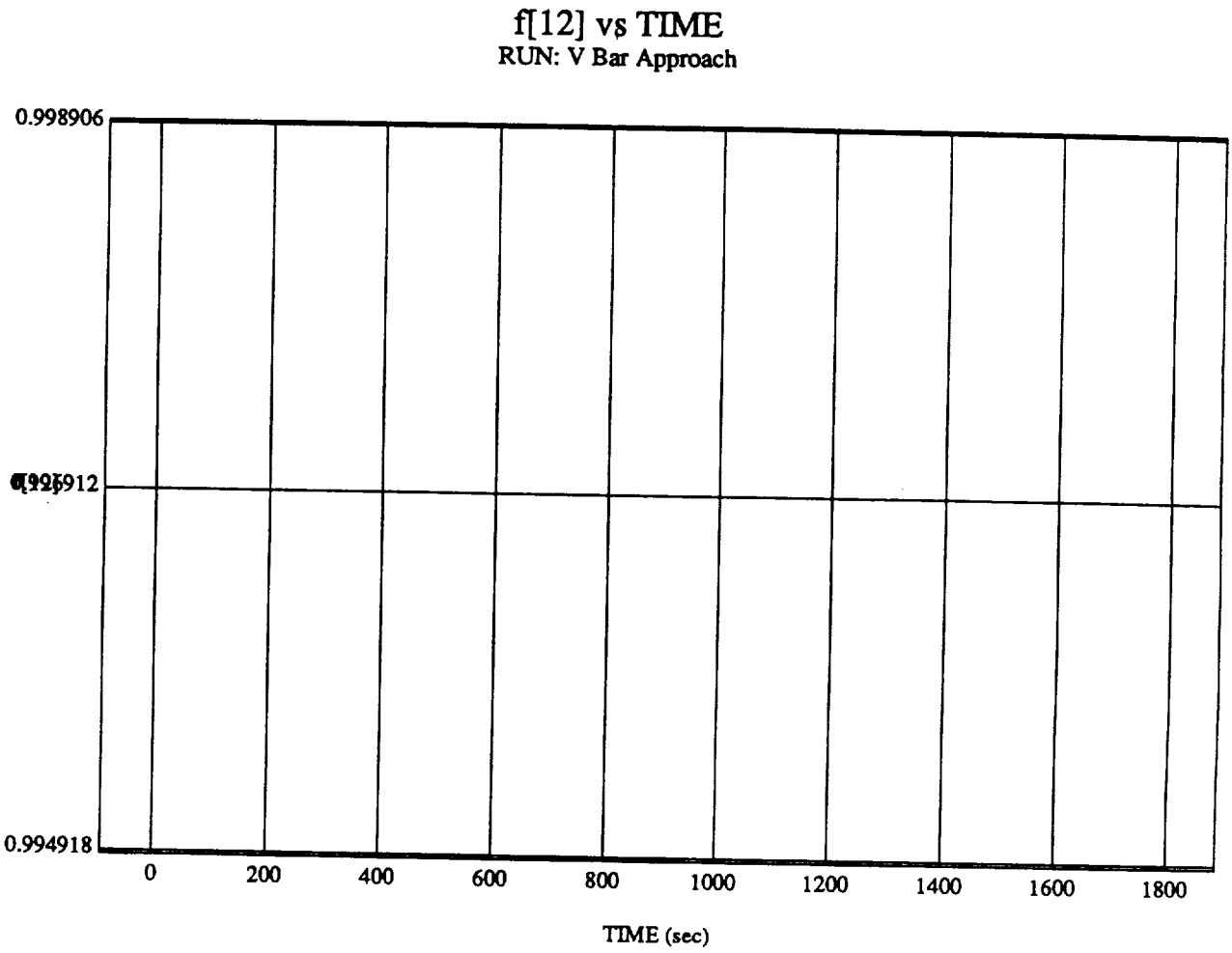
MODULE: ORBITER.lm_range
DATA SAMPLING FREQUENCY: 0.200 Hz

SIMULATION APPLICATION: ARIC Translational Controller Simulation

f[8] vs TIME
RUN: V Bar Approach

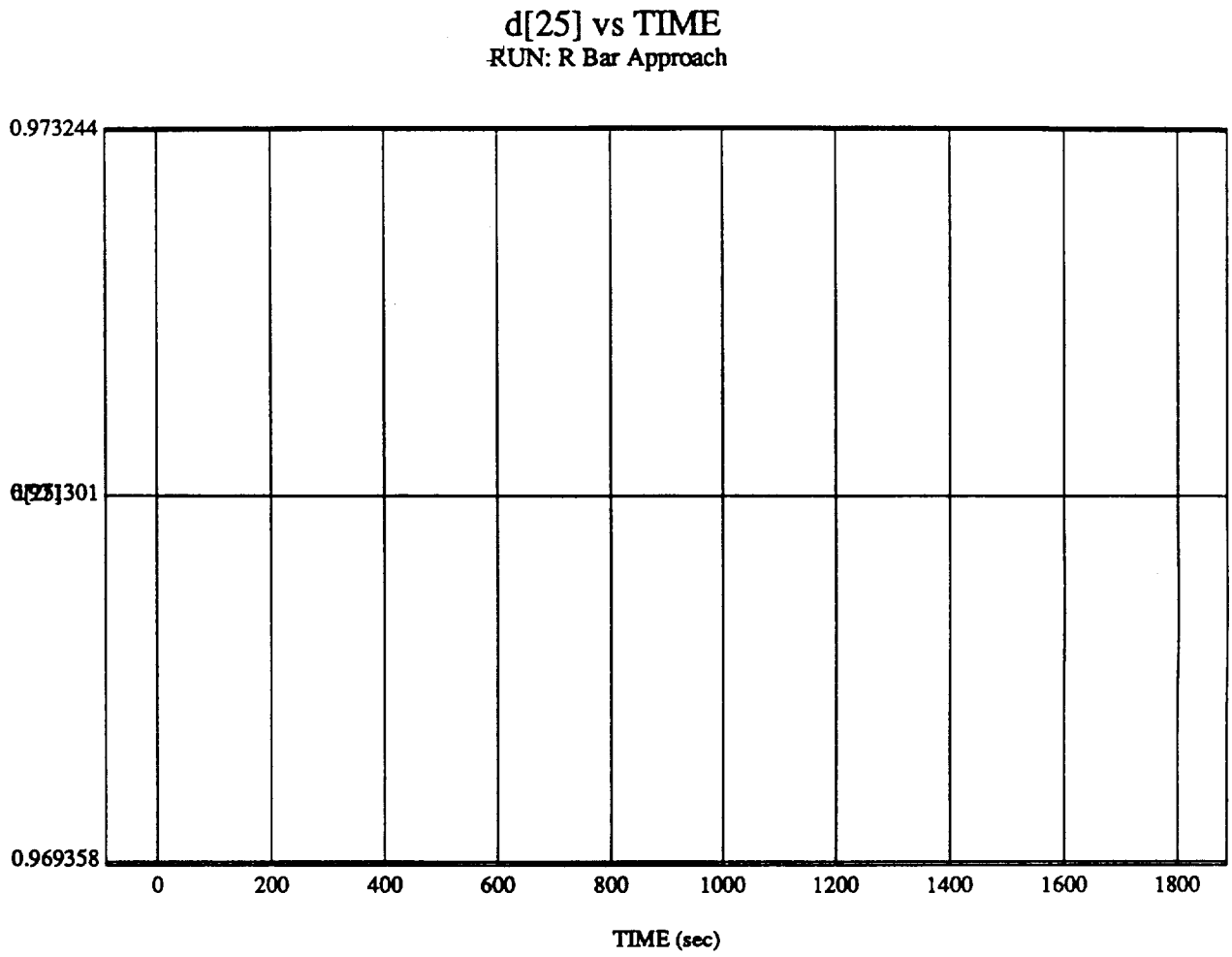


MODULE: ORBITER.lm_range
DATA SAMPLING FREQUENCY: 0.200 Hz

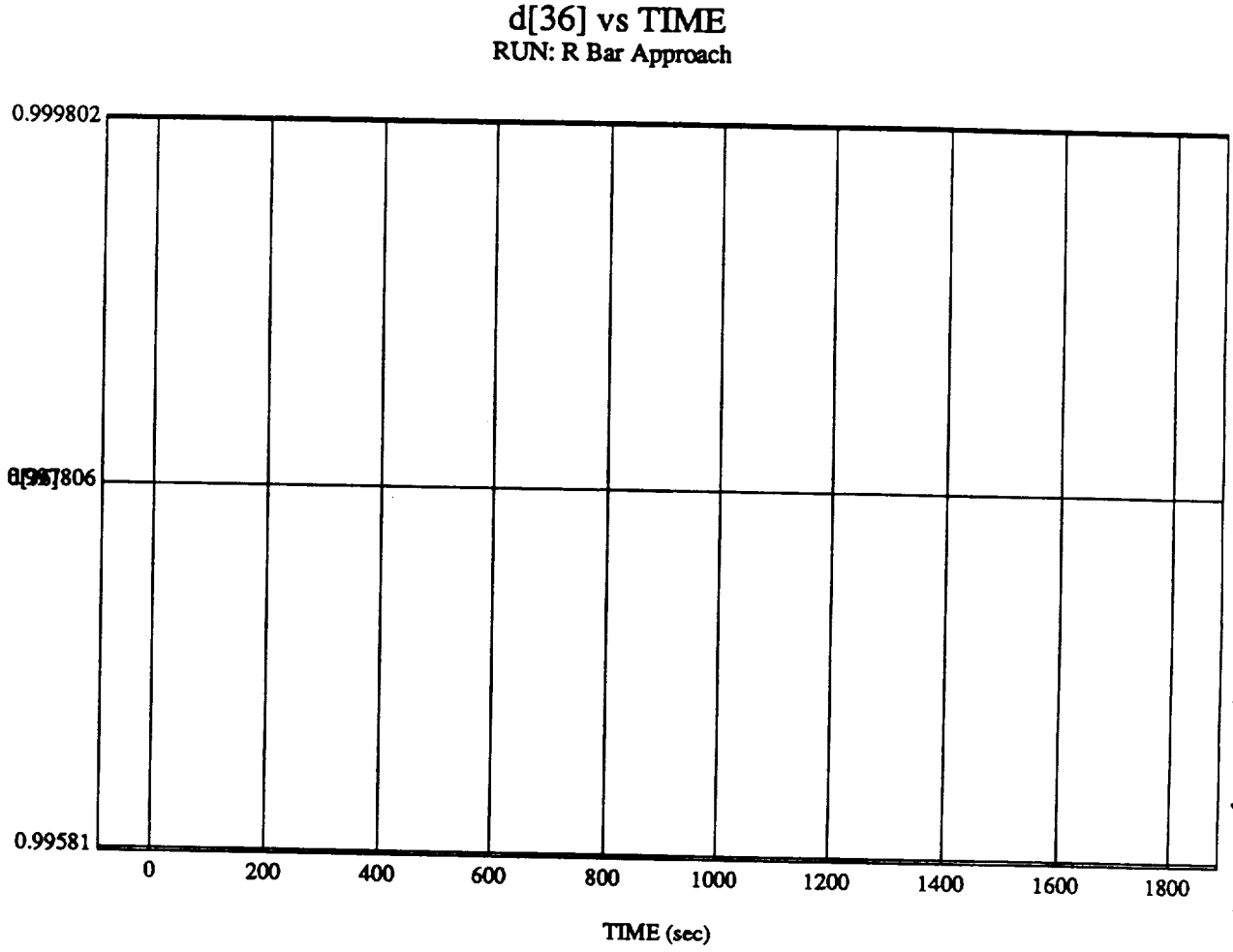


MODULE: ORBITER.lm_range
DATA SAMPLING FREQUENCY: 0.200 Hz

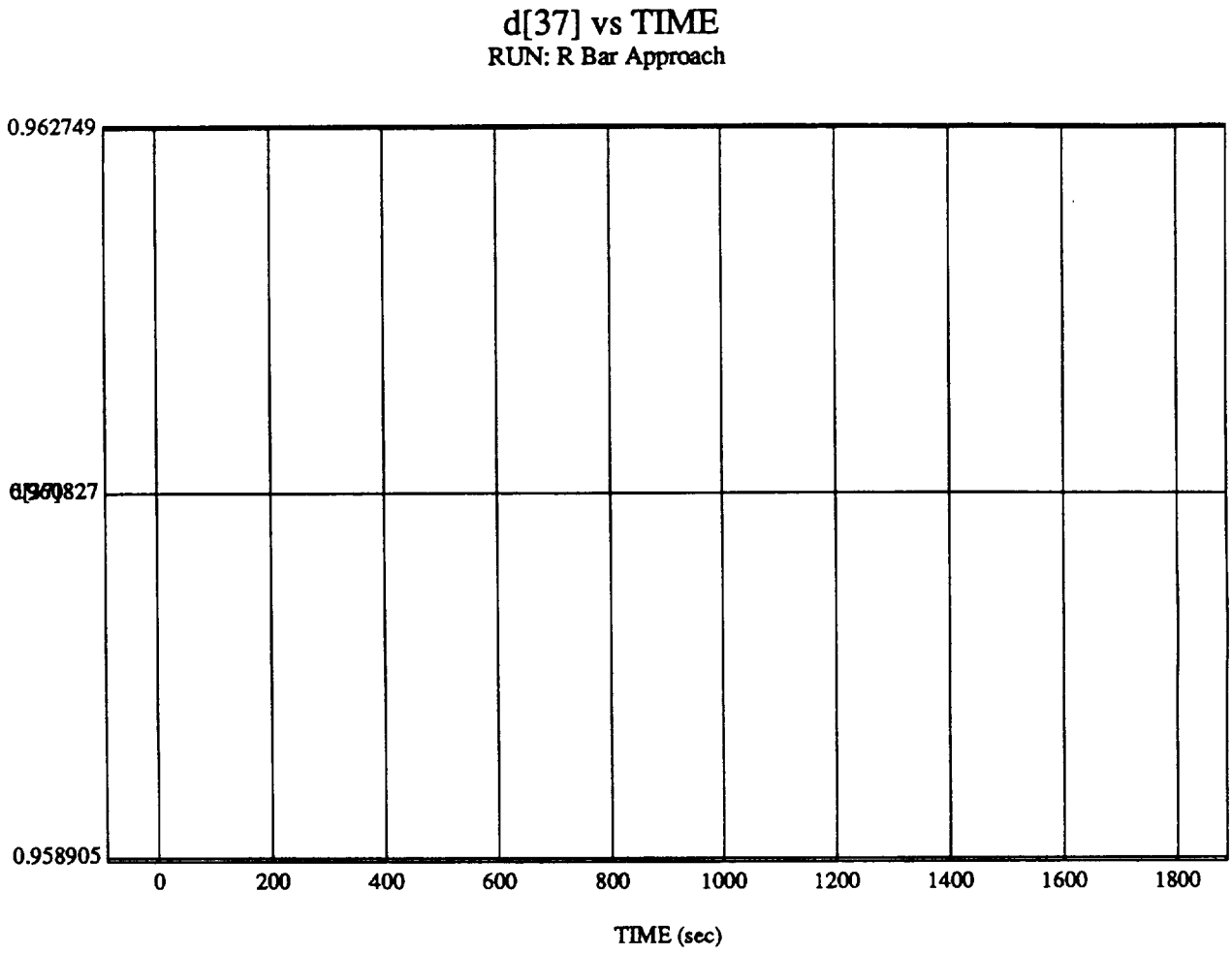
SIMULATION APPLICATION: ARIC Translational Controller Simulation



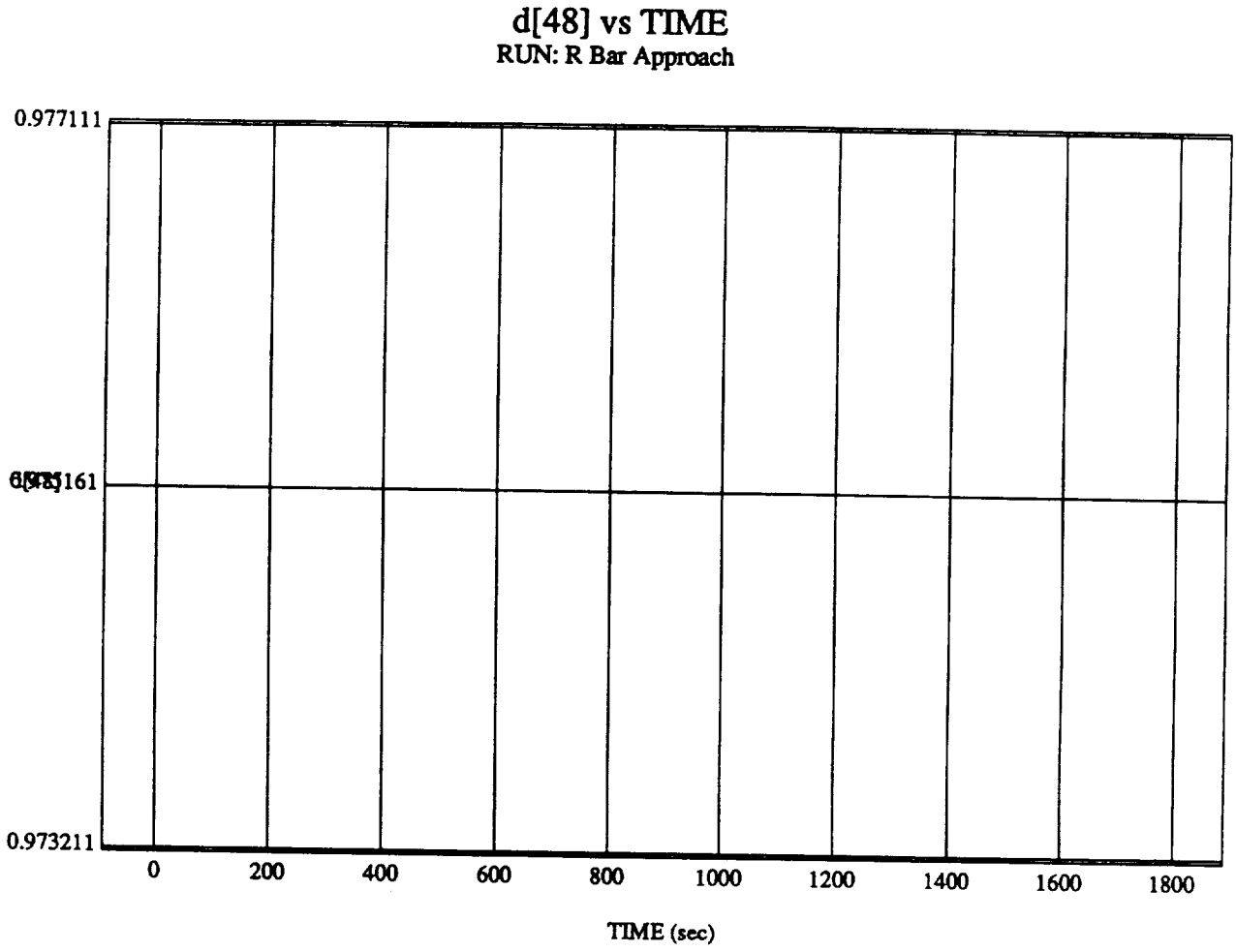
MODULE: ORBITER.lm_range
DATA SAMPLING FREQUENCY: 0.200 Hz



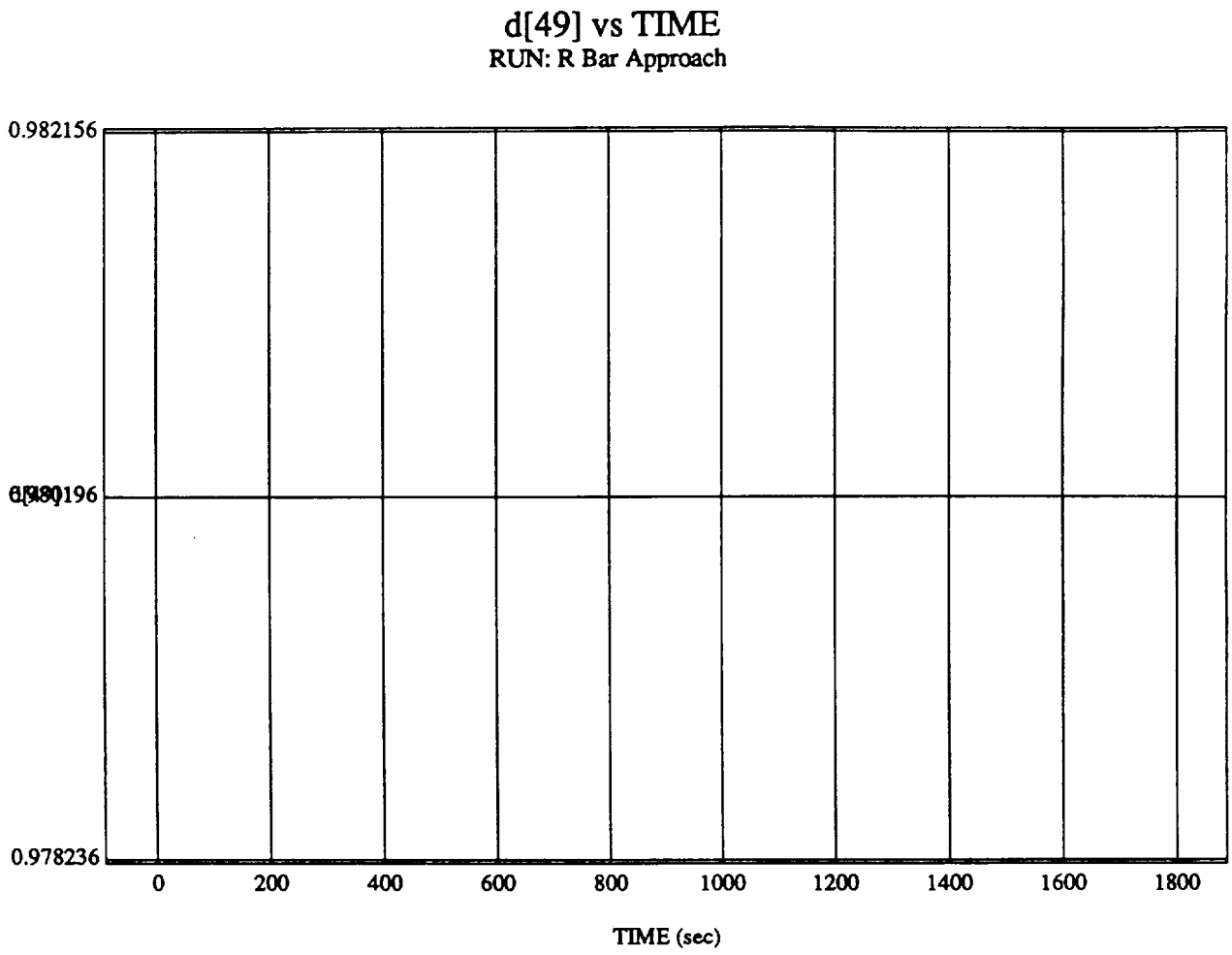
MODULE: ORBITER.lm_range
DATA SAMPLING FREQUENCY: 0.200 Hz



MODULE: ORBITER.lm_range
DATA SAMPLING FREQUENCY: 0.200 Hz



MODULE: ORBITER.lm_range
DATA SAMPLING FREQUENCY: 0.200 Hz

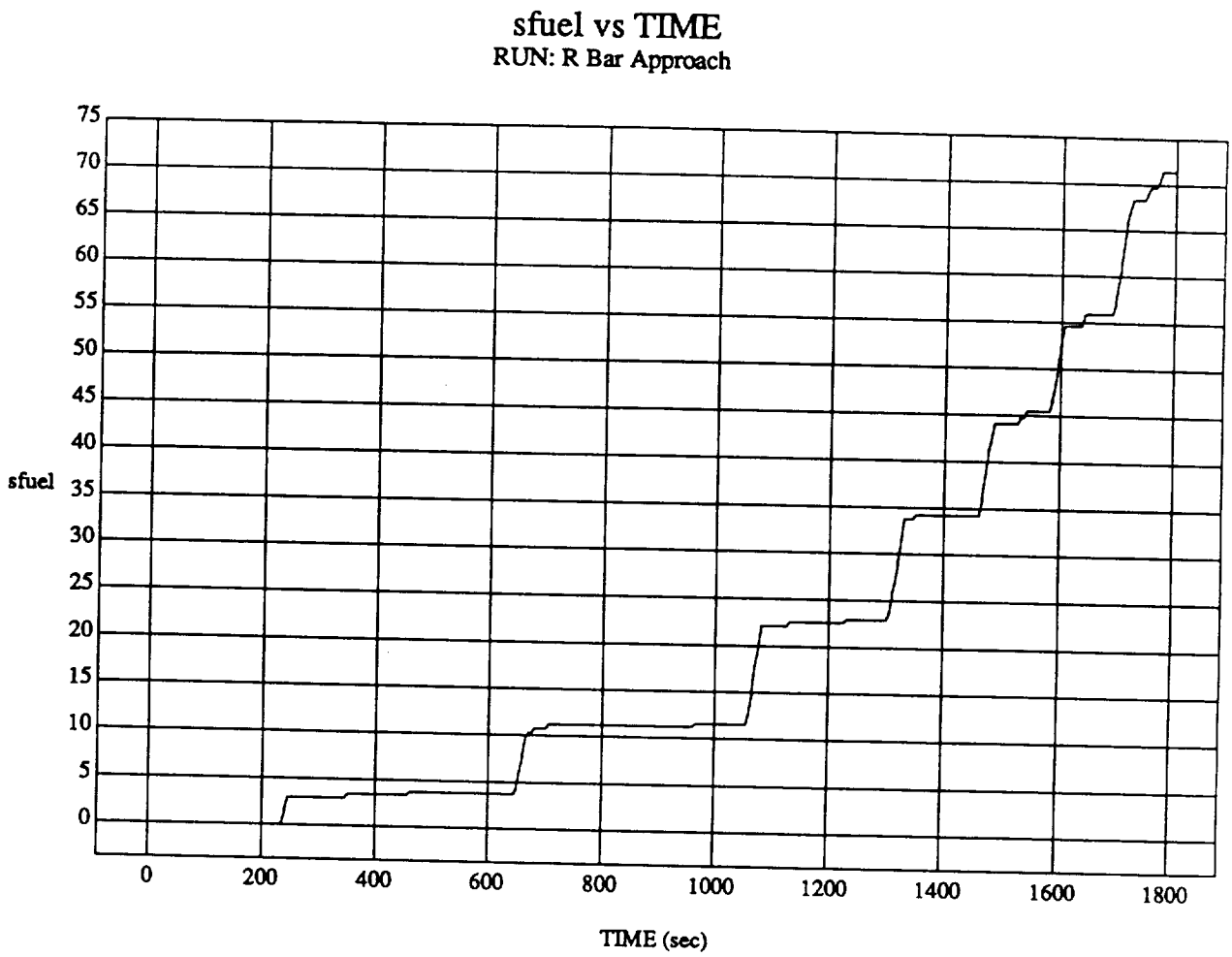


MODULE: ORBITER.lm_range
DATA SAMPLING FREQUENCY: 0.200 Hz

SIMULATION APPLICATION: ARIC Translational Controller Simulation

unusualness vs TIME
RUN: R Bar Approach

SIMULATION APPLICATION: ARIC Translational Controller Simulation



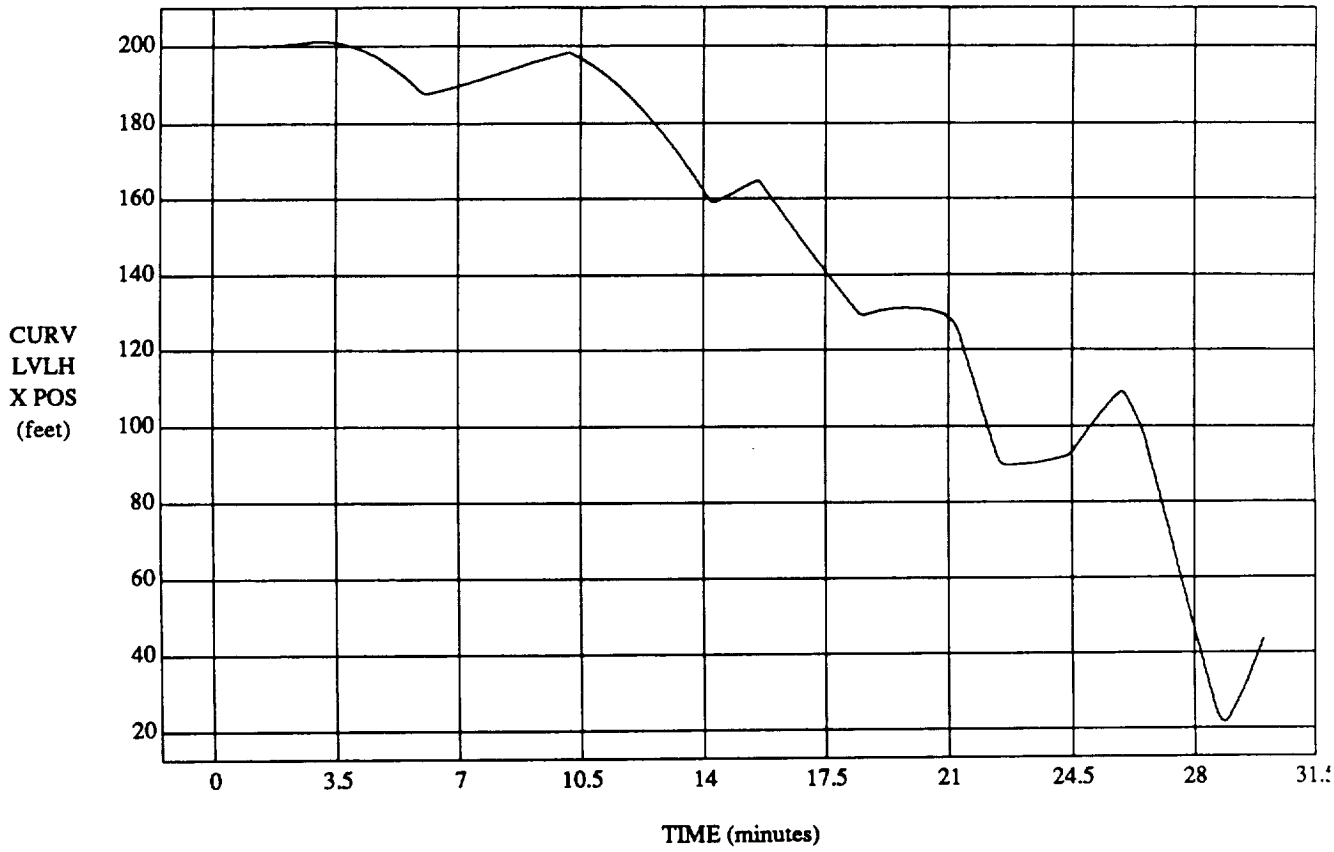
MODULE: ORBITER.primary
DATA SAMPLING FREQUENCY: 0.200 Hz

B3. Fly-Around from V-bar to R-bar vector in 30 minutes

SIMULATION APPLICATION: ARIC Translational Controller Simulation

TARGET CENTERED ROTATING CURVILINEAR LVLH X POS vs TIME

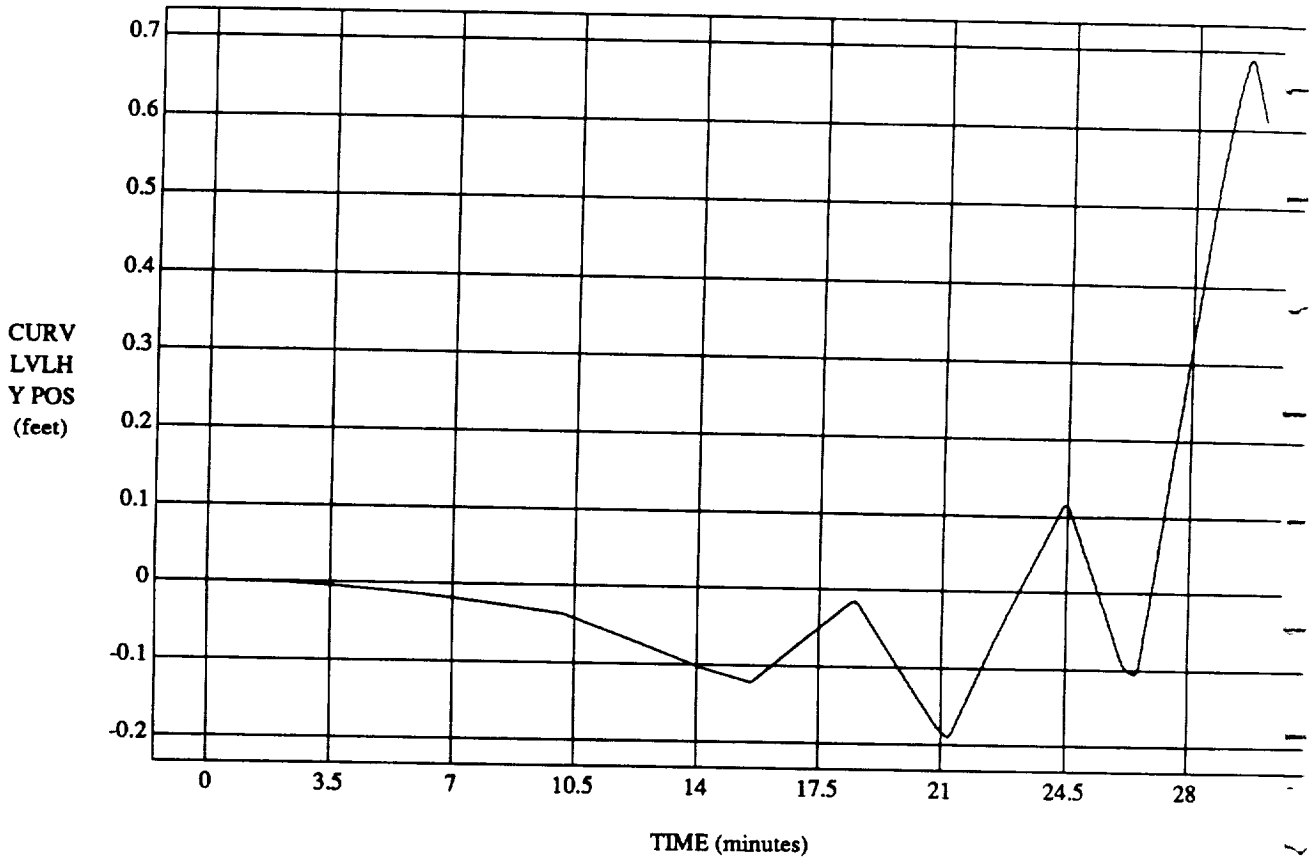
RUN: Fly Around - V Bar To -R Bar



VEHICLE: ORBITER.state
TARGET VEHICLE: SOLMAX.state
DATA SAMPLING FREQUENCY: 0.521 Hz

TARGET CENTERED ROTATING CURVILINEAR LVLH Y POS vs TIME

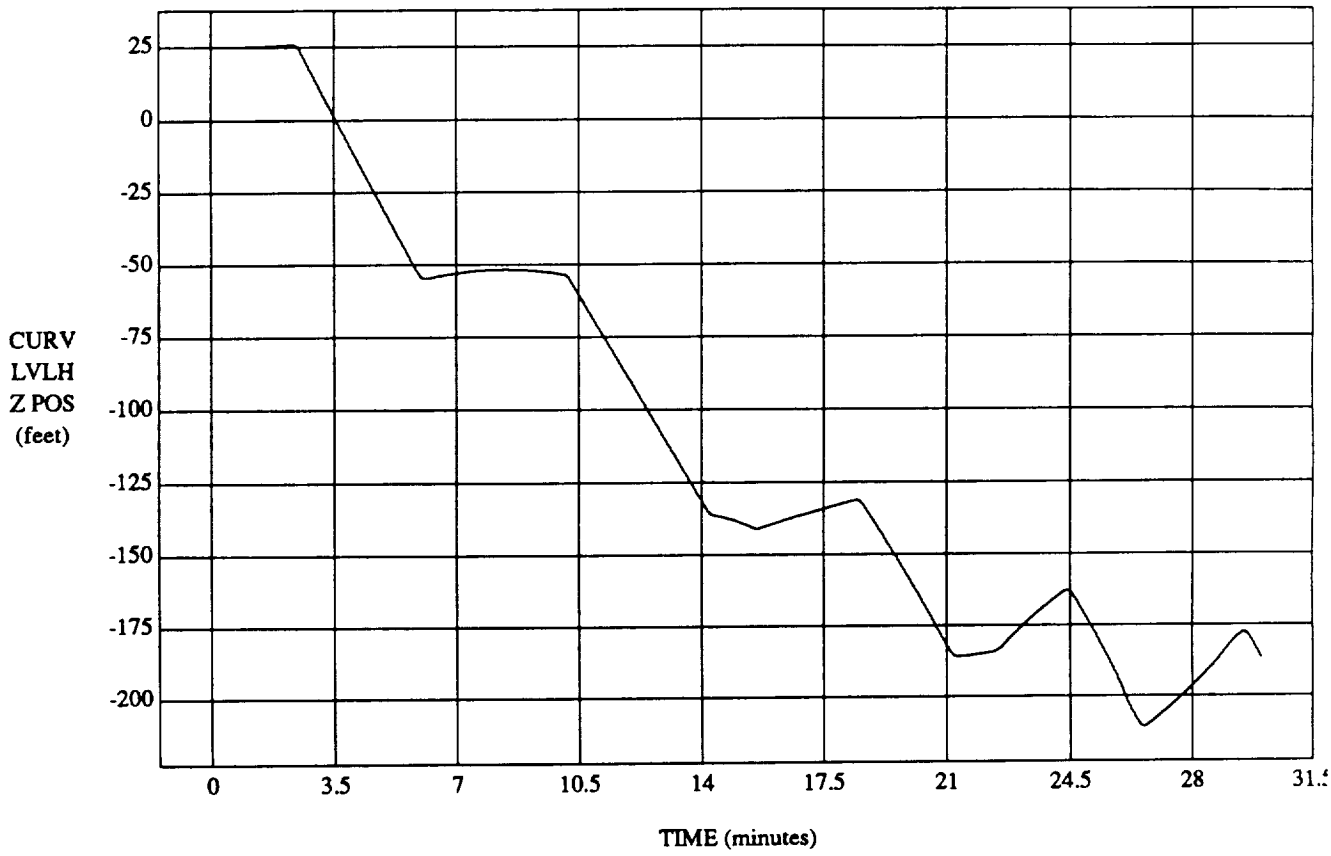
RUN: Fly Around - V Bar To -R Bar



VEHICLE: ORBITER.state
TARGET VEHICLE: SOLMAX.state
DATA SAMPLING FREQUENCY: 0.521 Hz

TARGET CENTERED ROTATING CURVILINEAR LVLH Z POS vs TIME

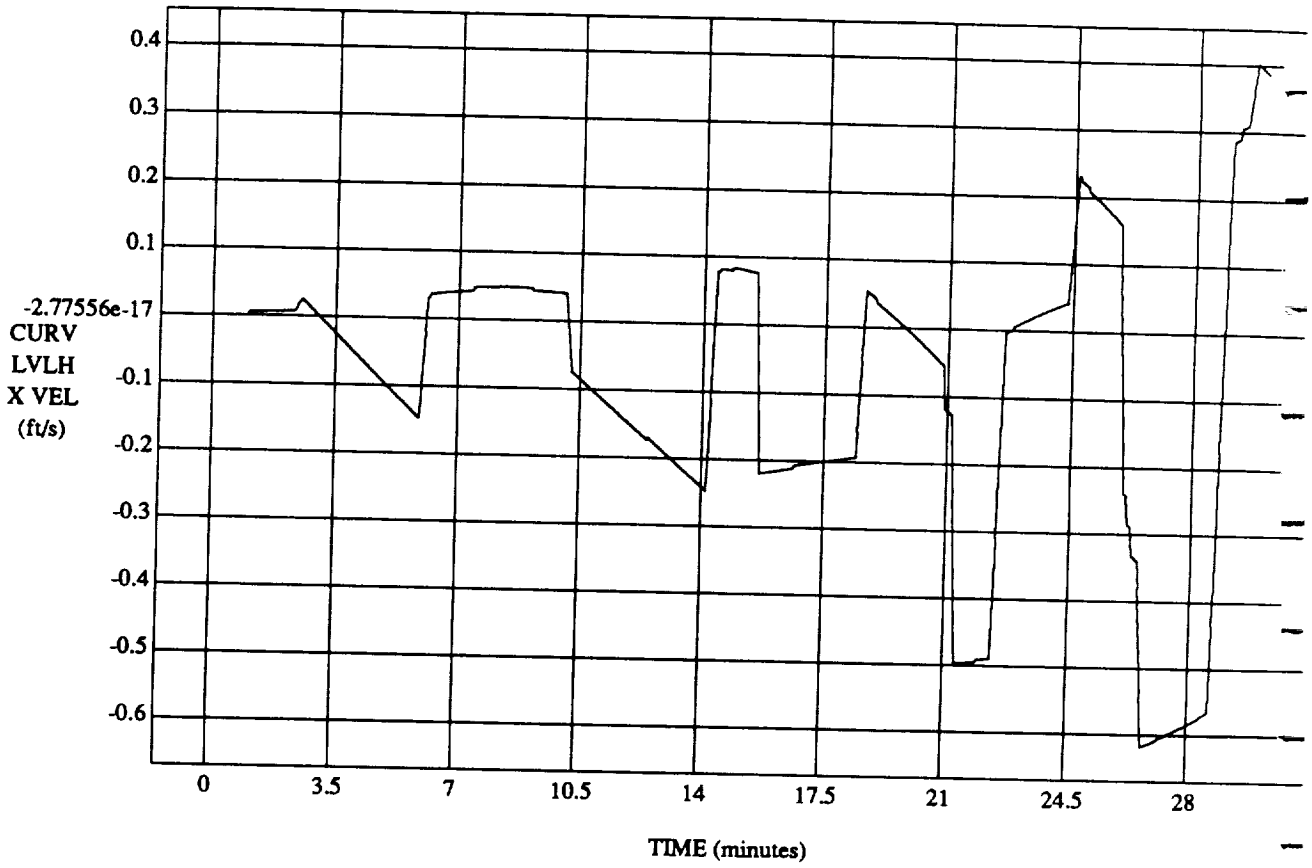
RUN: Fly Around - V Bar To -R Bar



VEHICLE: ORBITER.state
TARGET VEHICLE: SOLMAX.state
DATA SAMPLING FREQUENCY: 0.521 Hz

TARGET CENTERED ROTATING CURVILINEAR LVLH X VEL vs TIME

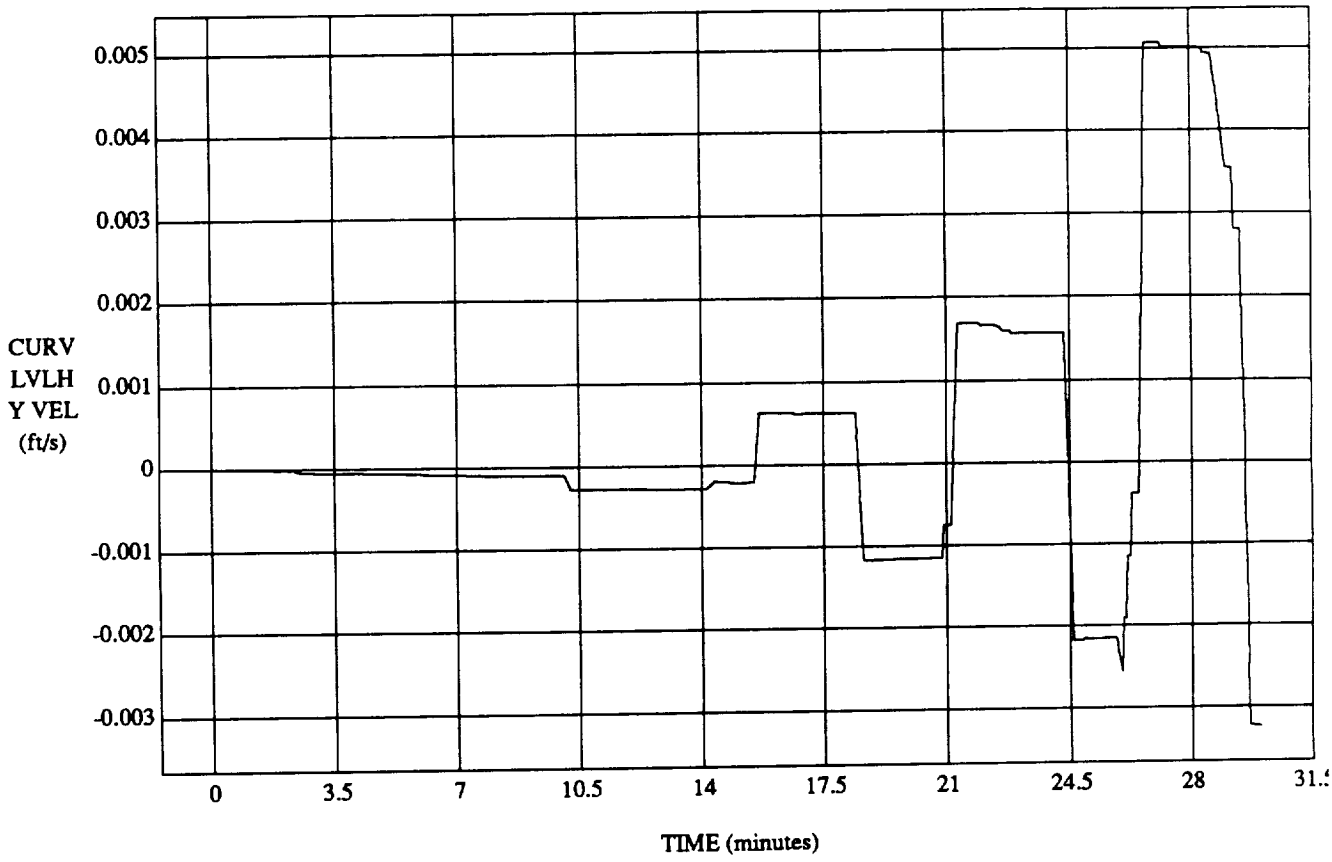
RUN: Fly Around - V Bar To -R Bar



VEHICLE: ORBITER.state
TARGET VEHICLE: SOLMAX.state
DATA SAMPLING FREQUENCY: 0.521 Hz

TARGET CENTERED ROTATING CURVILINEAR LVLH Y VEL vs TIME

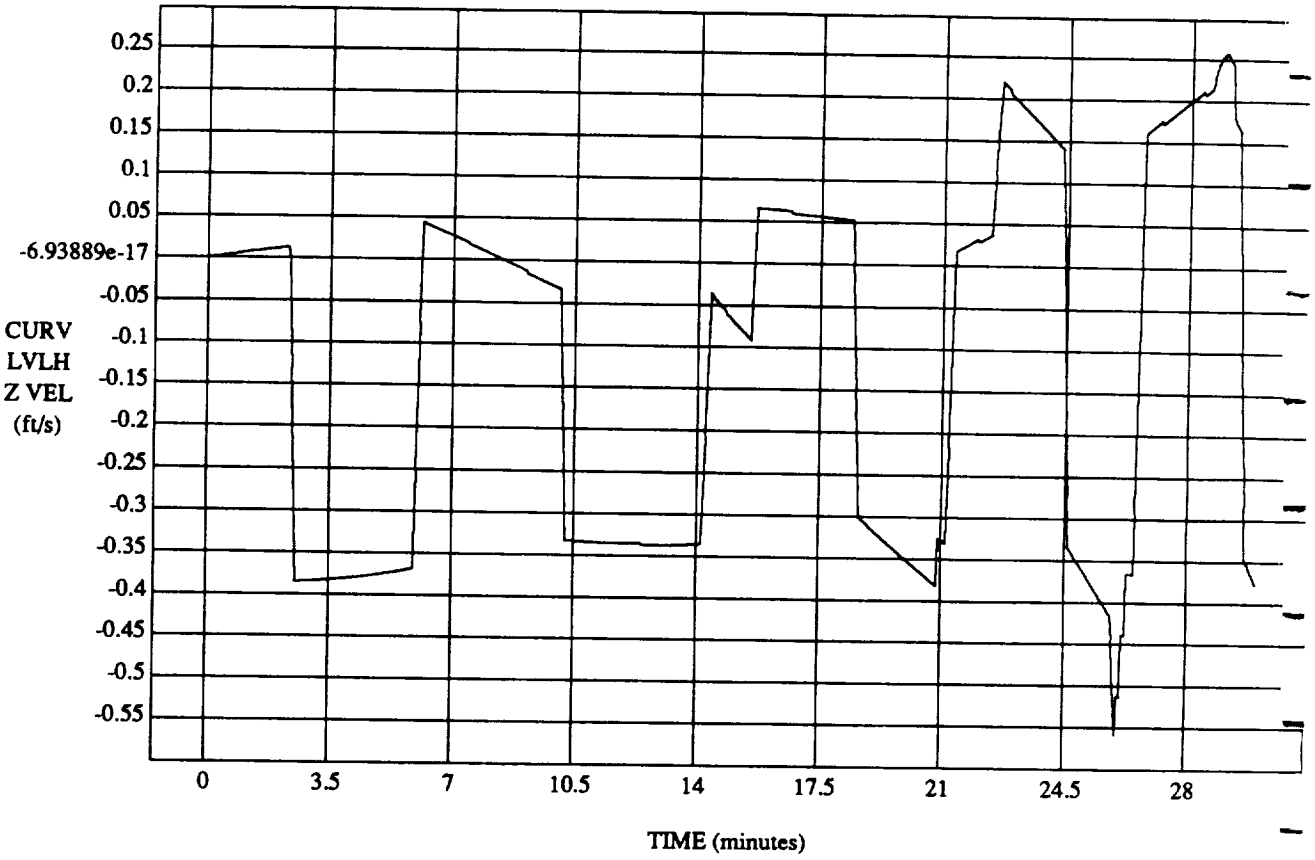
RUN: Fly Around - V Bar To -R Bar



VEHICLE: ORBITER.state
TARGET VEHICLE: SOLMAX.state
DATA SAMPLING FREQUENCY: 0.521 Hz

TARGET CENTERED ROTATING CURVILINEAR LVLH Z VEL vs TIME

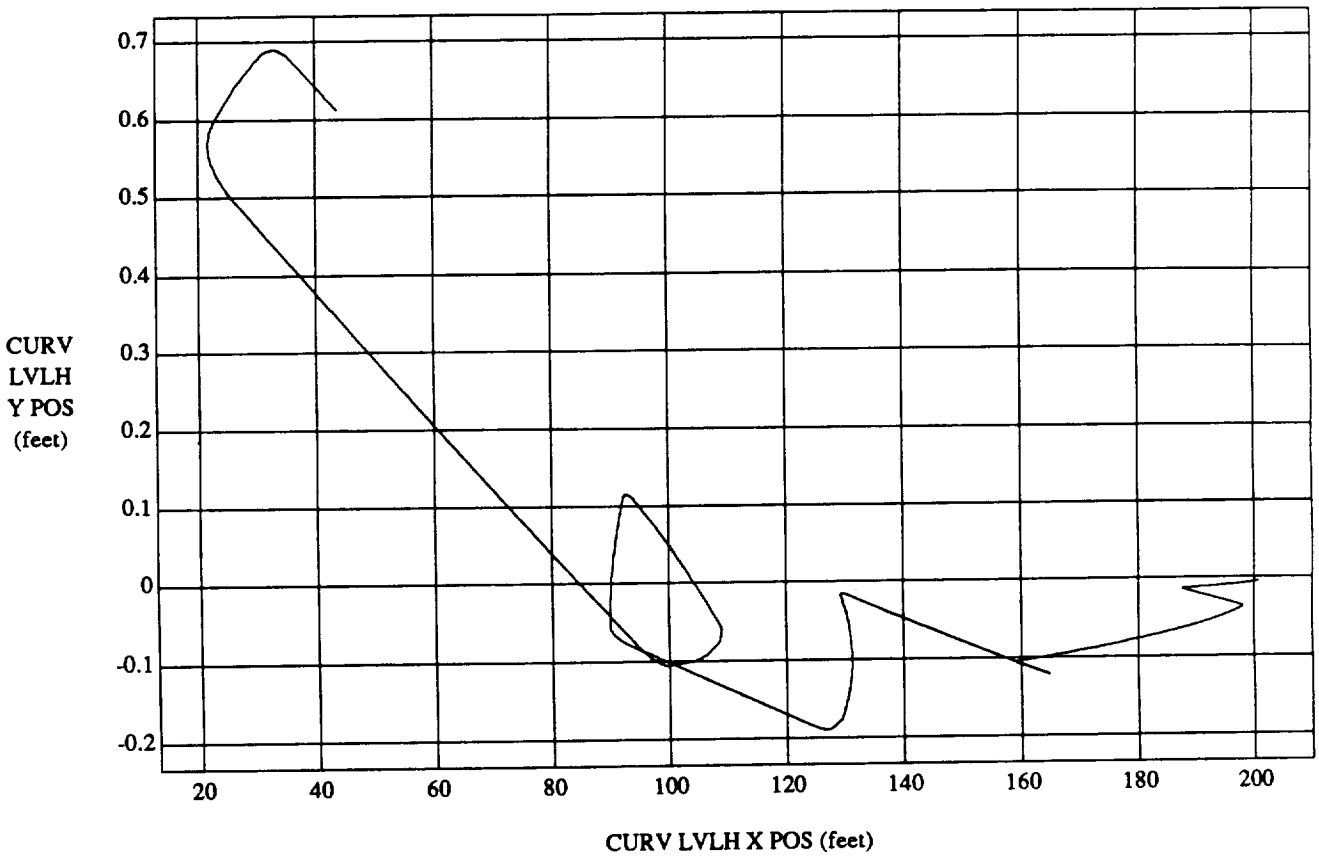
RUN: Fly Around - V Bar To -R Bar



VEHICLE: ORBITER.state
TARGET VEHICLE: SOLMAX.state
DATA SAMPLING FREQUENCY: 0.521 Hz

TARGET CENTERED ROTATING CURVILINEAR LVLH Y POS vs X POS

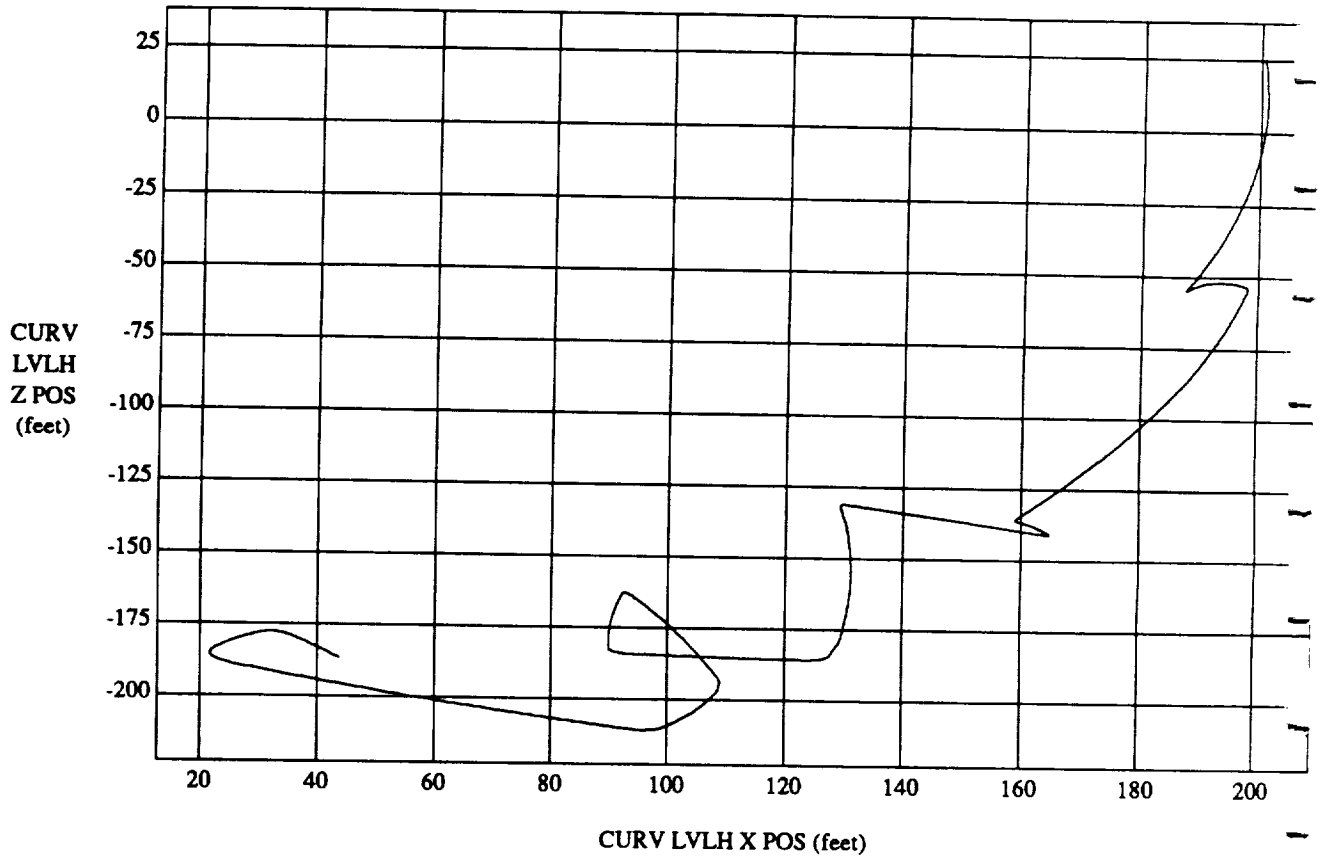
RUN: Fly Around - V Bar To -R Bar



VEHICLE: ORBITER.state
TARGET VEHICLE: SOLMAX.state
DATA SAMPLING FREQUENCY: 0.521 Hz

TARGET CENTERED ROTATING CURVILINEAR LVLH Z POS vs X POS

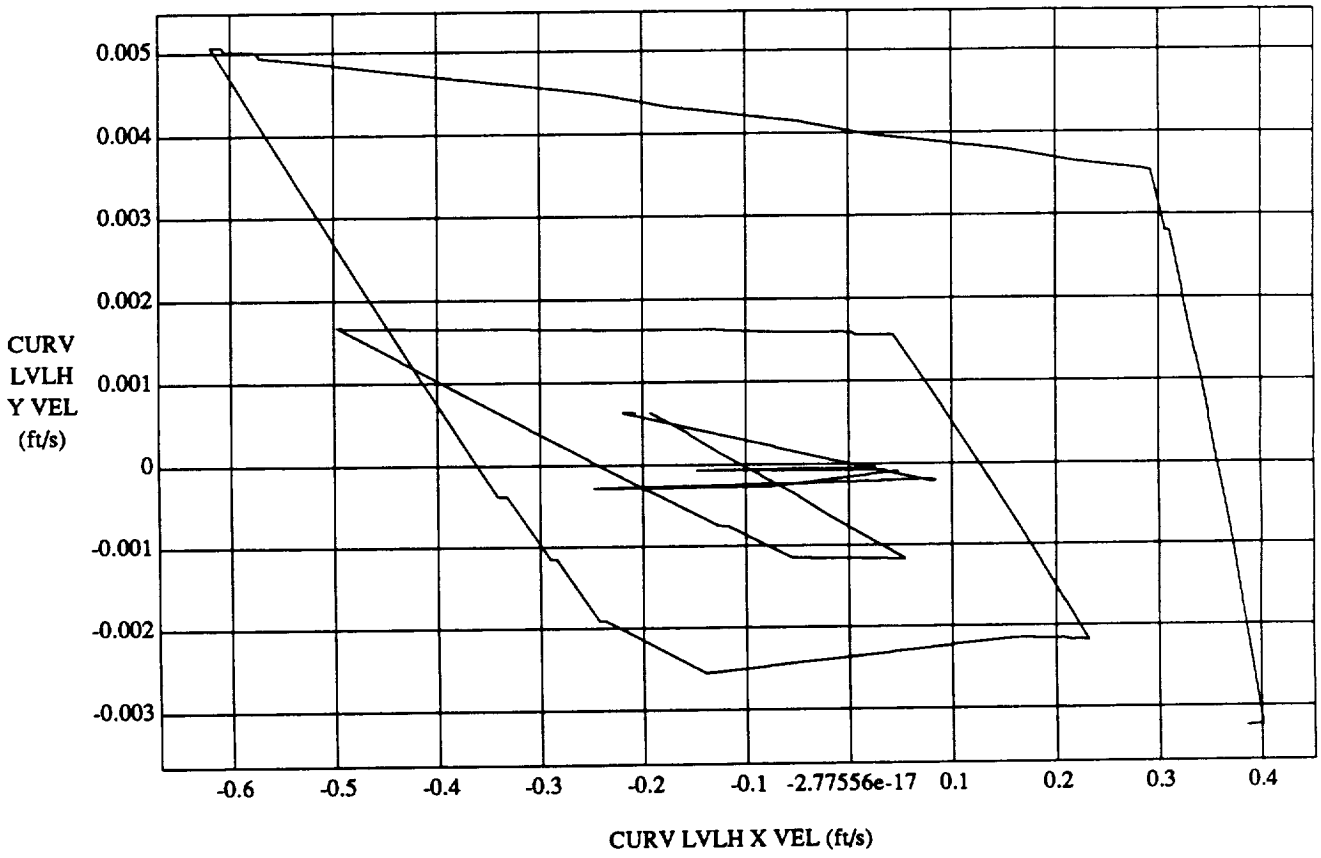
RUN: Fly Around - V Bar To -R Bar



VEHICLE: ORBITER.state
TARGET VEHICLE: SOLMAX.state
DATA SAMPLING FREQUENCY: 0.521 Hz

TARGET CENTERED ROTATING CURVILINEAR LVLH Y VEL vs X VEL

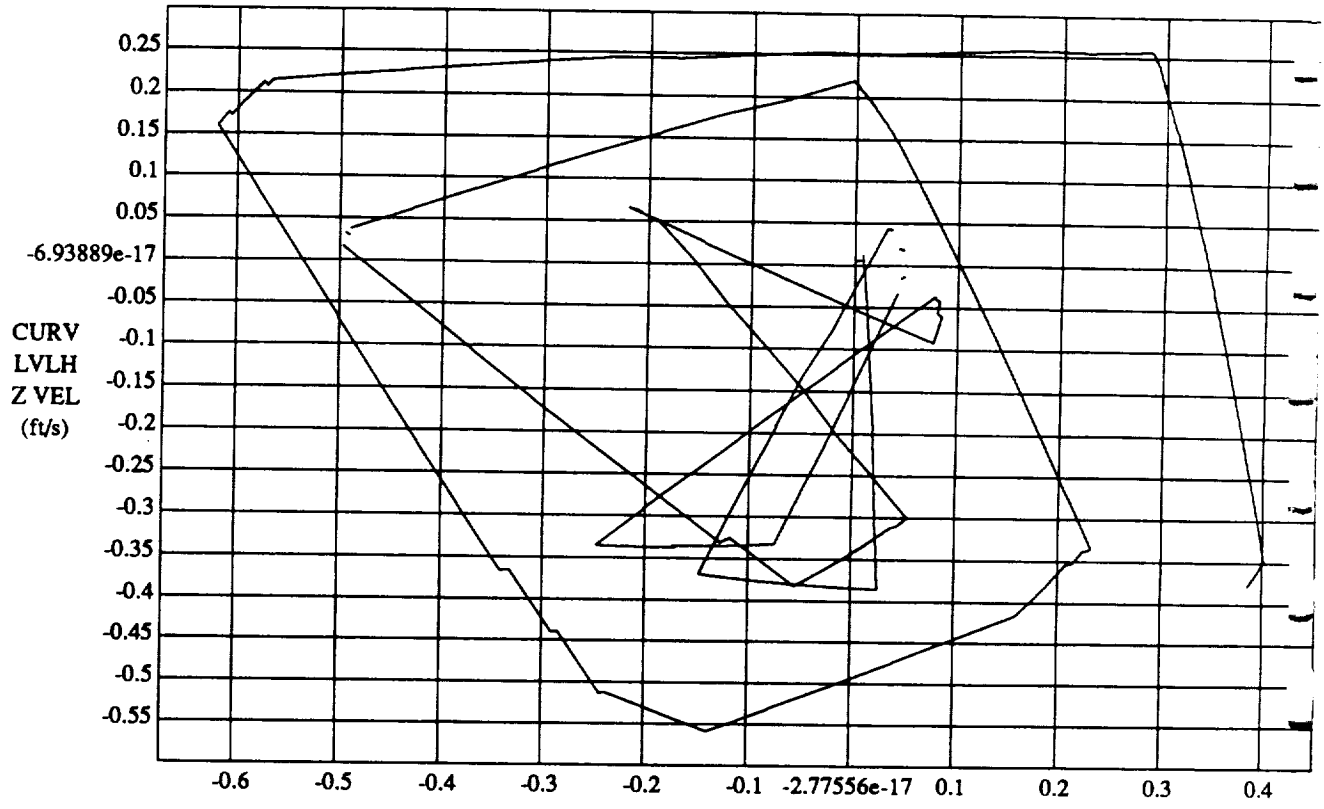
RUN: Fly Around - V Bar To -R Bar



VEHICLE: ORBITER.state
TARGET VEHICLE: SOLMAX.state
DATA SAMPLING FREQUENCY: 0.521 Hz

TARGET CENTERED ROTATING CURVILINEAR LVLH Z VEL vs X VEL

RUN: Fly Around - V Bar To -R Bar

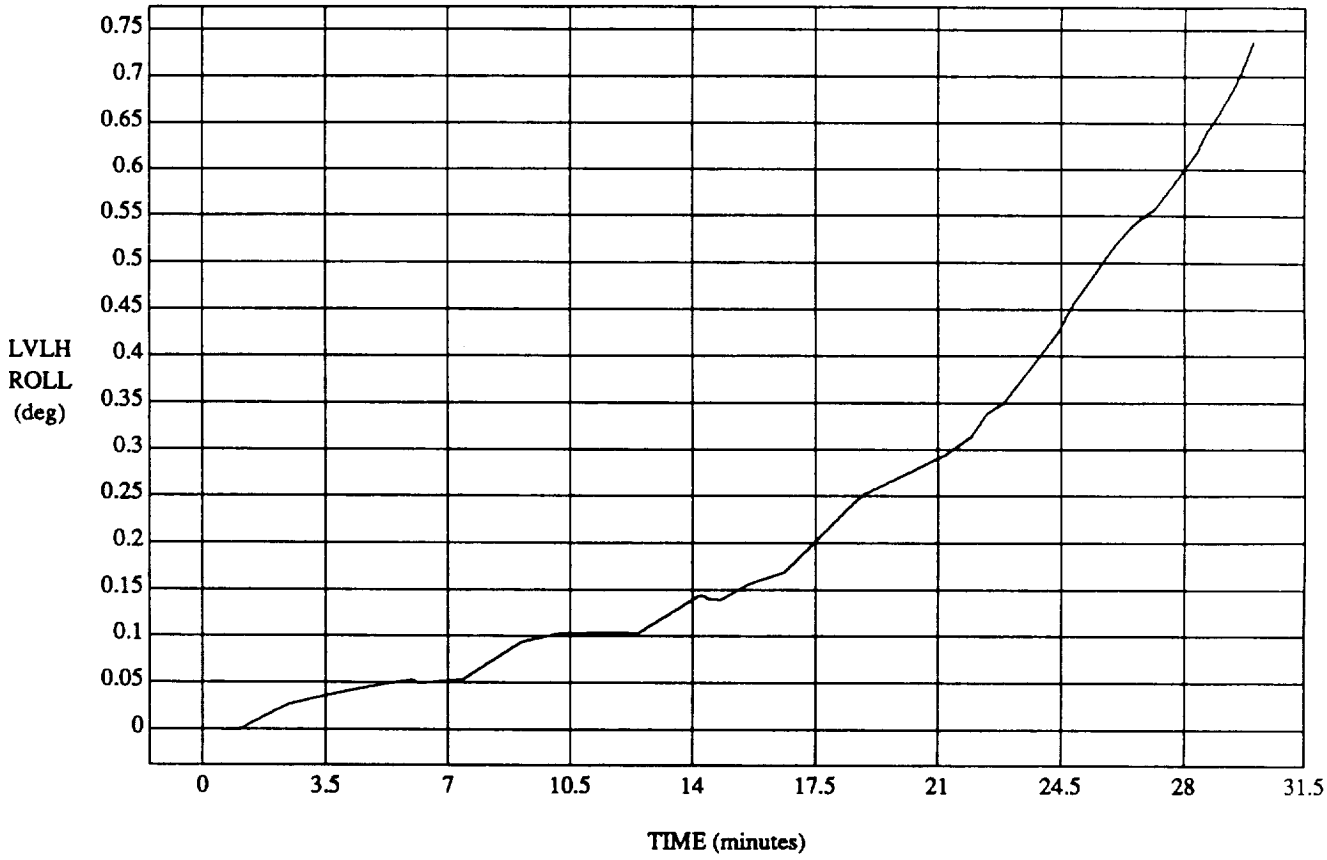


VEHICLE: ORBITER.state
TARGET VEHICLE: SOLMAX.state
DATA SAMPLING FREQUENCY: 0.521 Hz

SIMULATION APPLICATION: ARIC Translational Controller Simulation

LVLH EULER PYR ROLL vs TIME

RUN: Fly Around - V Bar To -R Bar

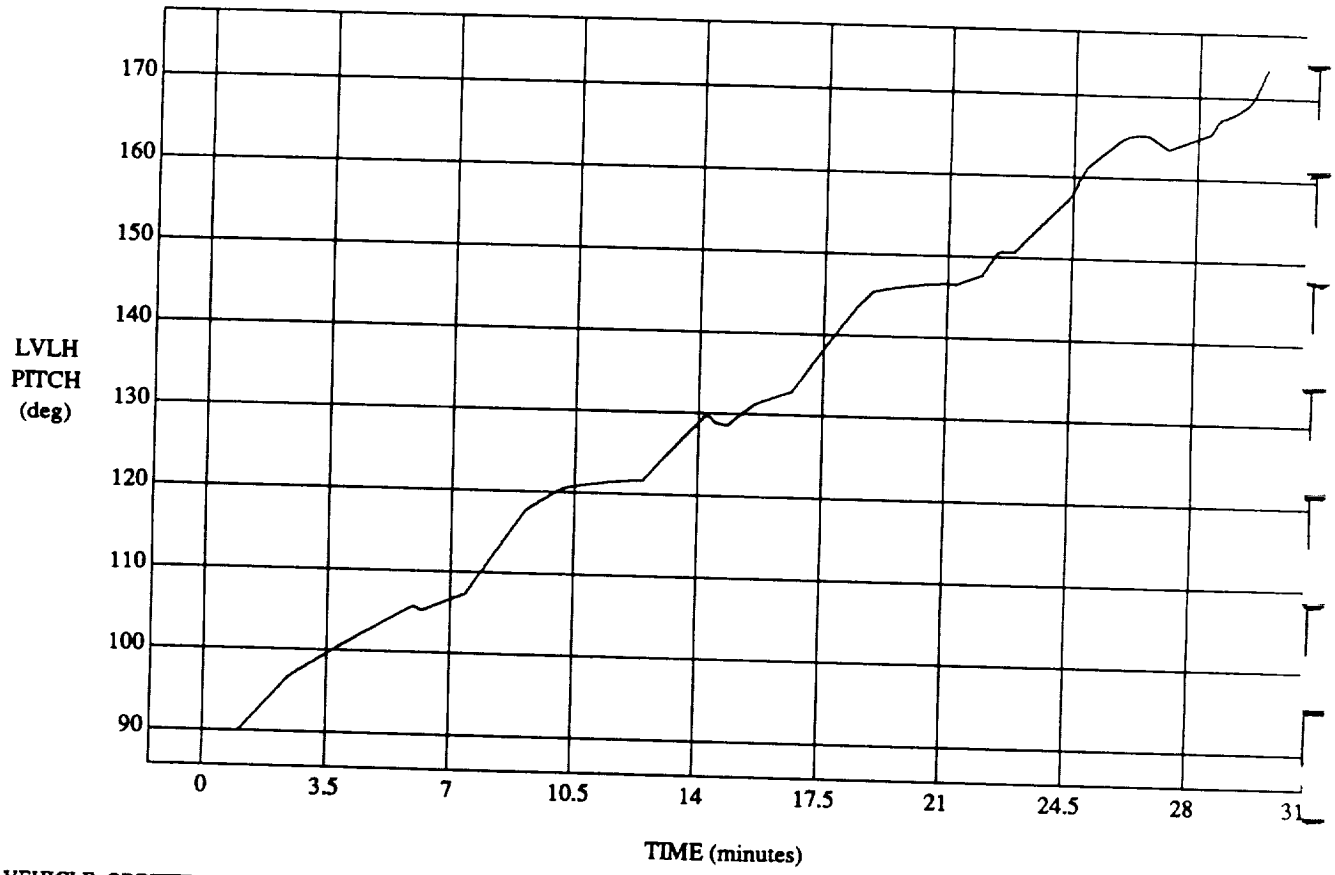


VEHICLE: ORBITER.state

DATA SAMPLING FREQUENCY: 0.521 Hz

LVLH EULER PYR PITCH vs TIME

RUN: Fly Around - V Bar To -R Bar

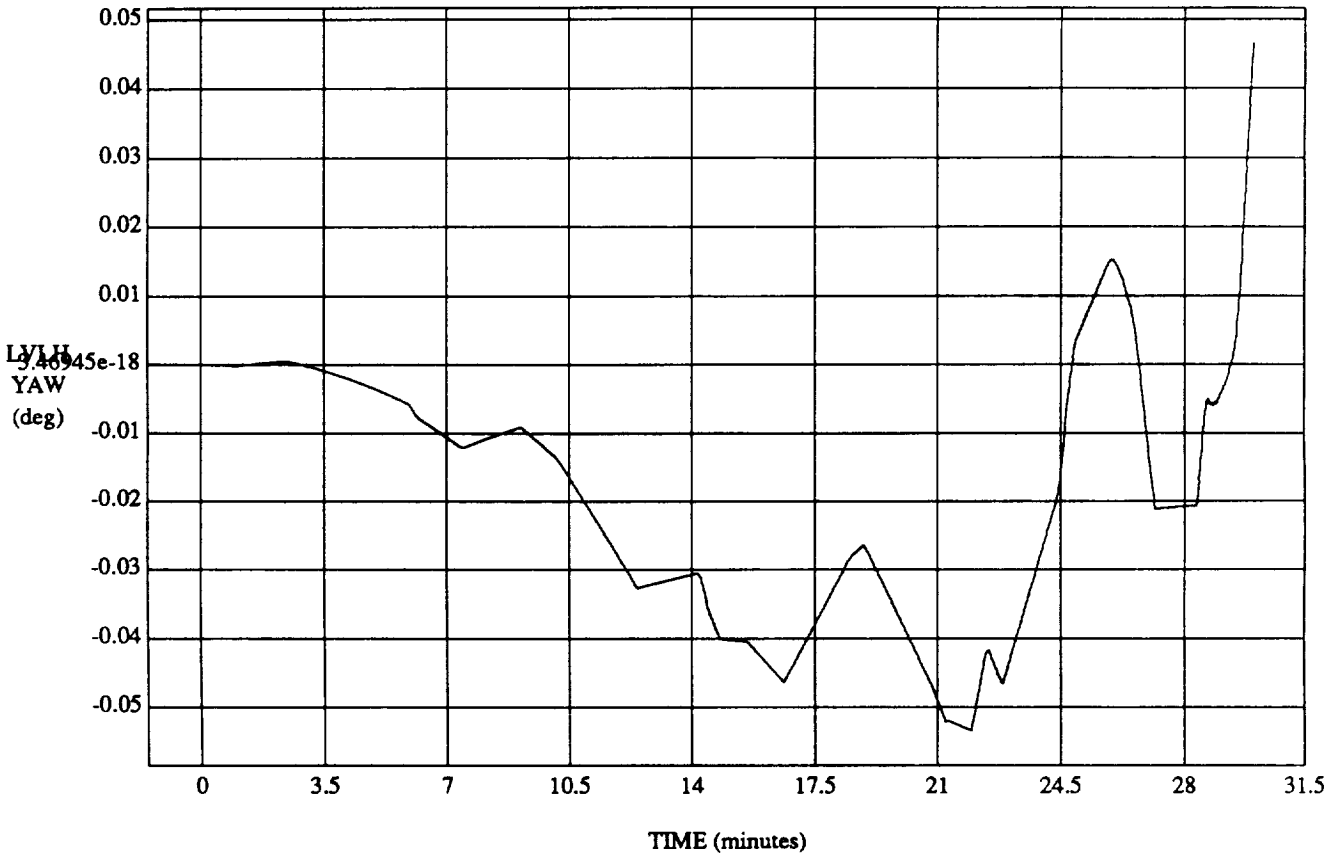


VEHICLE: ORBITER.state

DATA SAMPLING FREQUENCY: 0.521 Hz

LVLH EULER PYR YAW vs TIME

RUN: Fly Around - V Bar To -R Bar

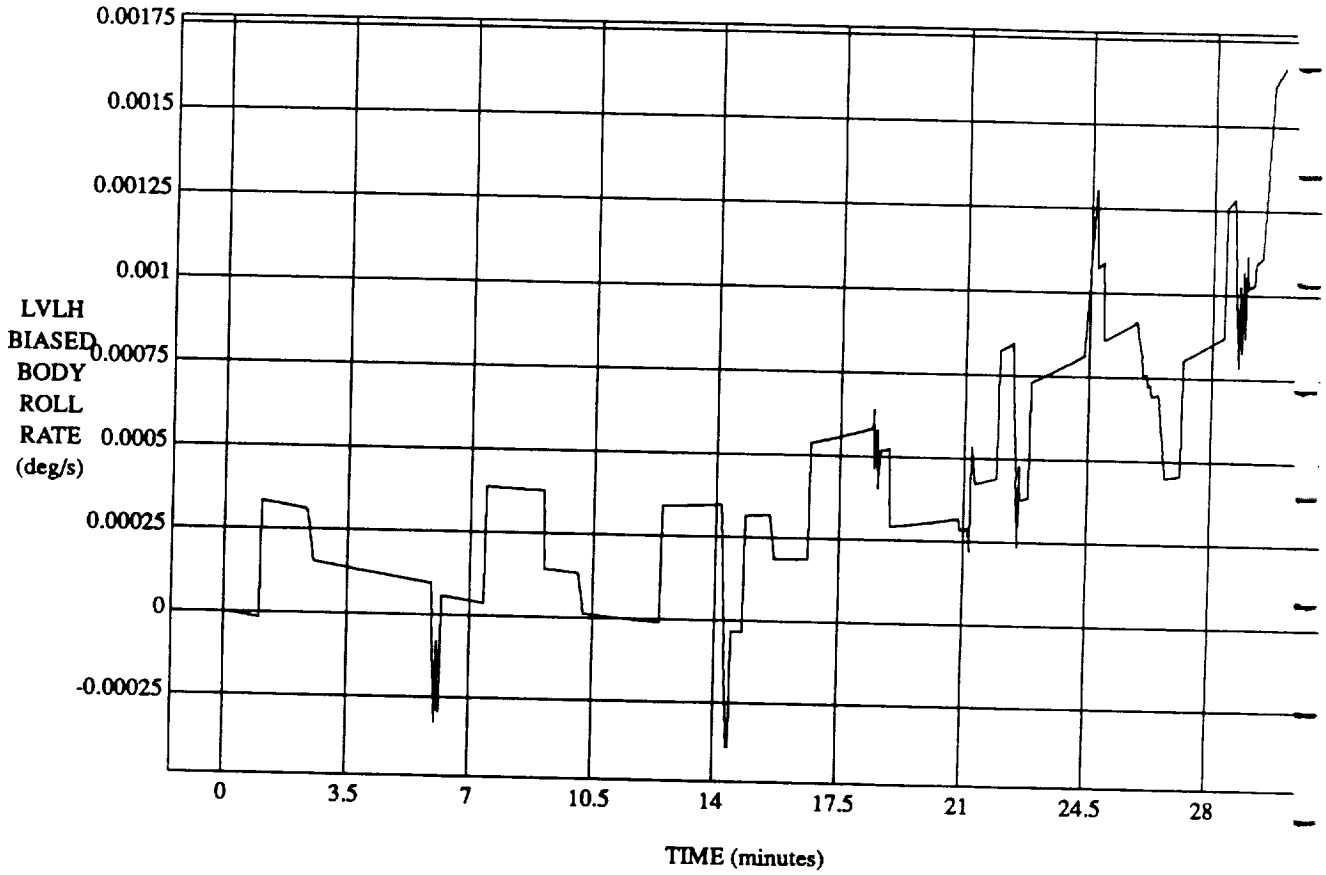


VEHICLE: ORBITER.state

DATA SAMPLING FREQUENCY: 0.521 Hz

LVLH BIASED BODY ROLL RATE vs TIME

RUN: Fly Around - V Bar To -R Bar

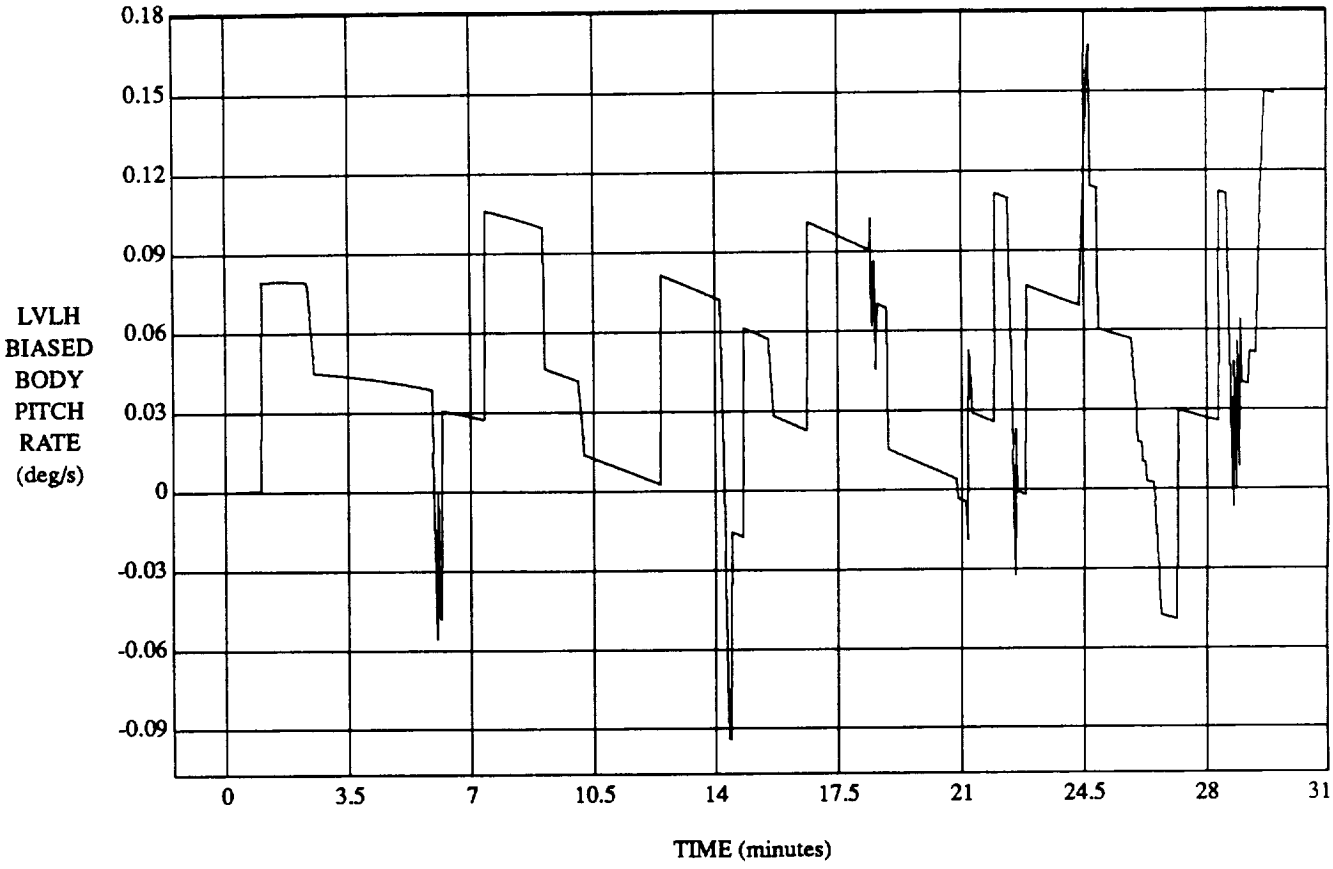


VEHICLE: ORBITER.state

DATA SAMPLING FREQUENCY: 0.521 Hz

LVLH BIASED BODY PITCH RATE vs TIME

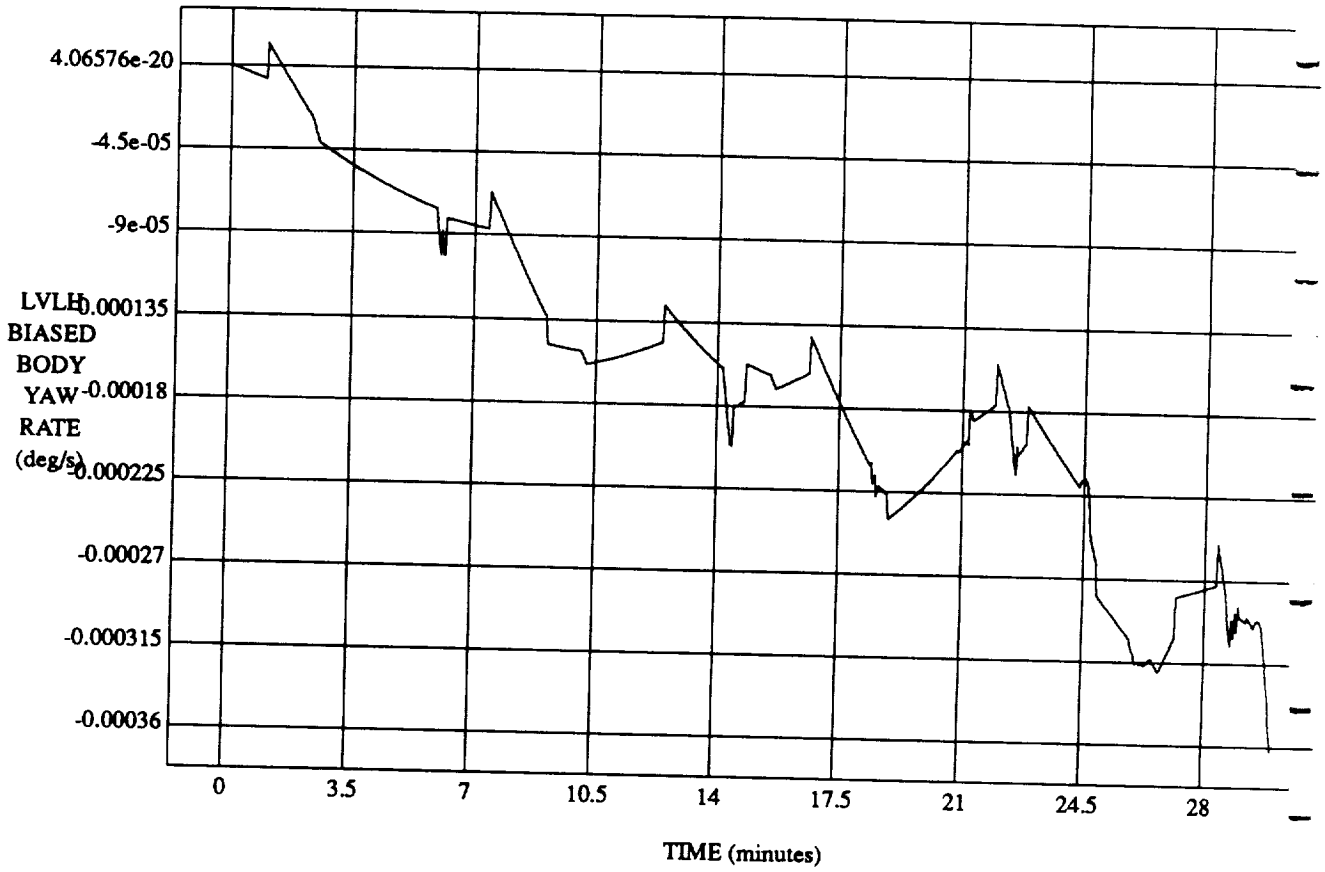
RUN: Fly Around - V Bar To -R Bar



VEHICLE: ORBITER.state
DATA SAMPLING FREQUENCY: 0.521 Hz

LVLH BIASED BODY YAW RATE vs TIME

RUN: Fly Around - V Bar To -R Bar



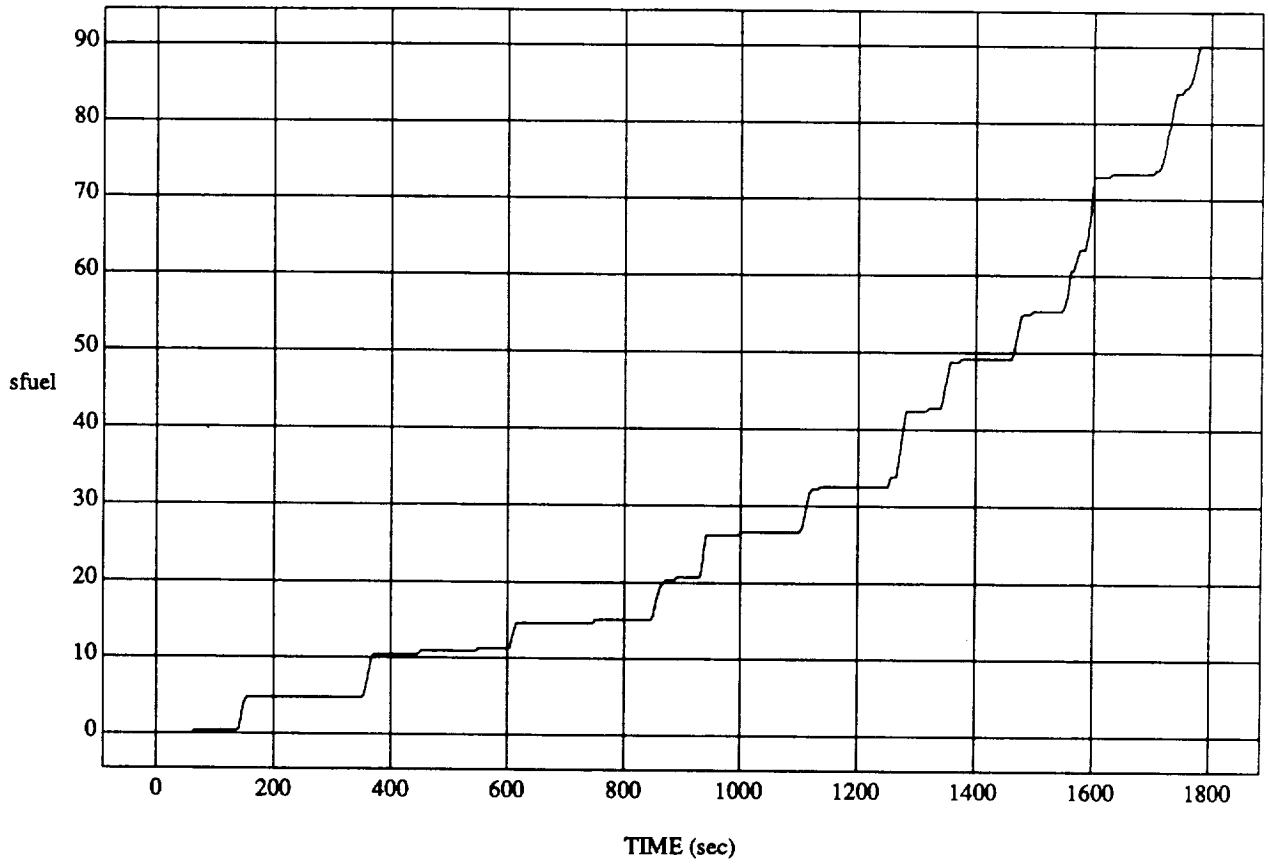
VEHICLE: ORBITER.state

DATA SAMPLING FREQUENCY: 0.521 Hz

SIMULATION APPLICATION: ARIC Translational Controller Simulation

sfuel vs TIME

RUN: Fly Around - V Bar To -R Bar



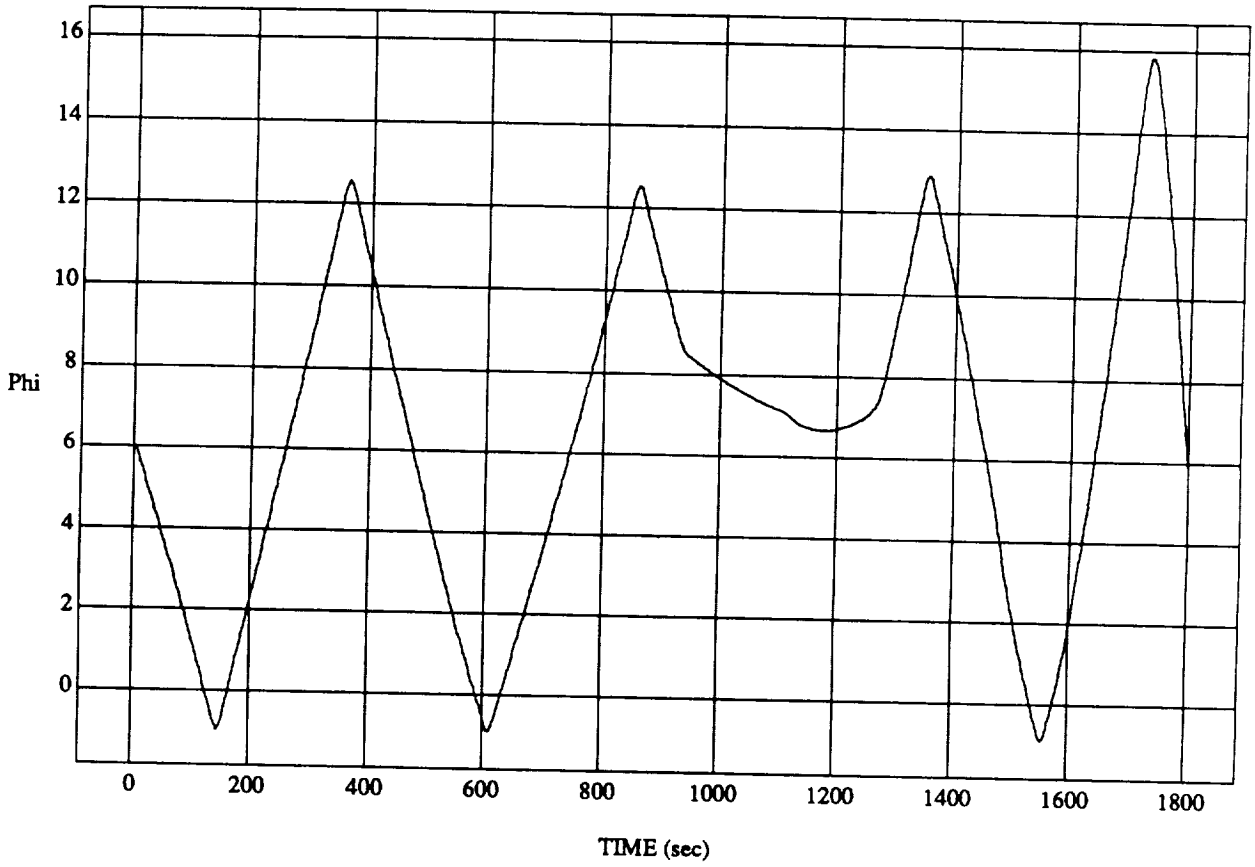
MODULE: ORBITER.primary

DATA SAMPLING FREQUENCY: 0.200 Hz

SIMULATION APPLICATION: ARIC Translational Controller Simulation

Phi vs TIME

RUN: Fly Around - V Bar To -R Bar

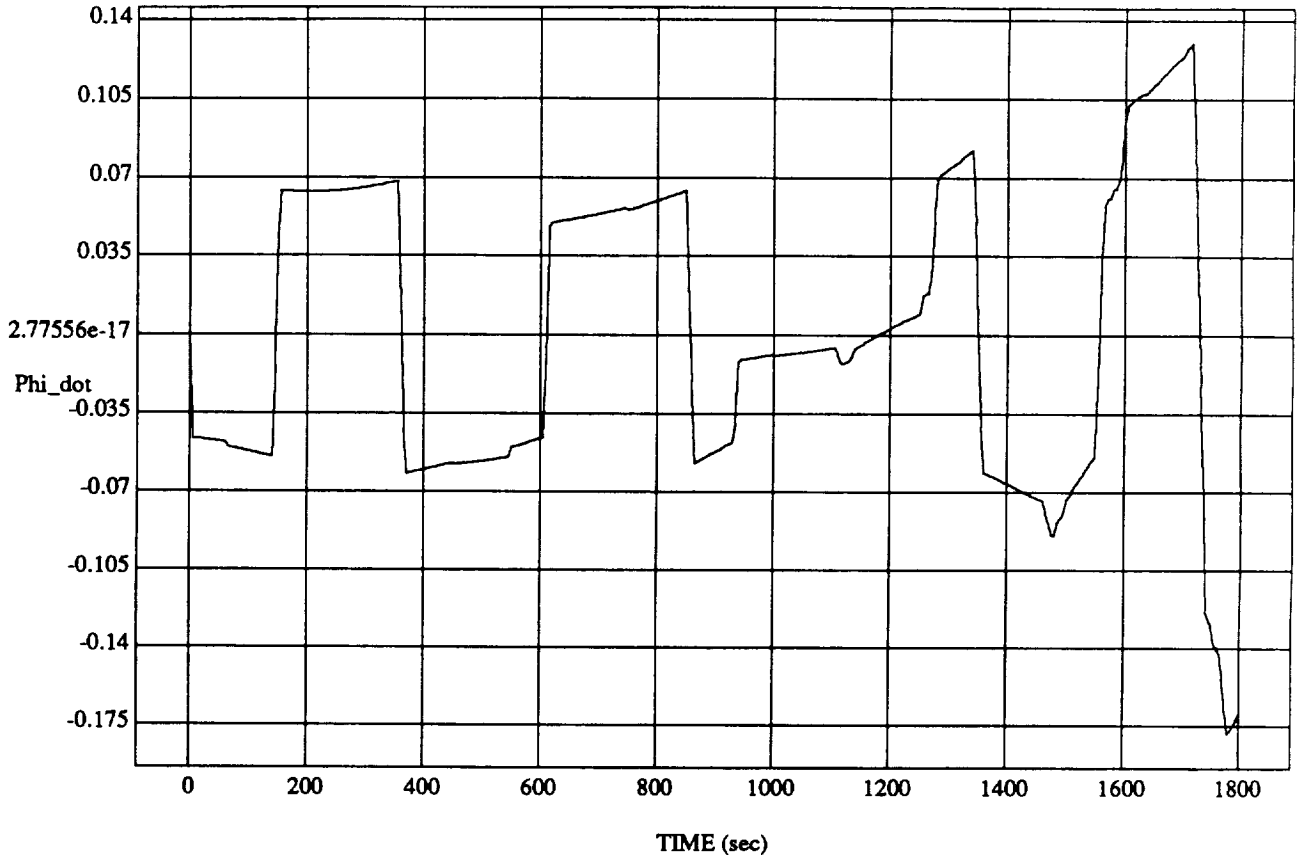


MODULE: ORBITER.lm_elev

DATA SAMPLING FREQUENCY: 0.200 Hz

Phi_dot vs TIME

RUN: Fly Around - V Bar To -R Bar

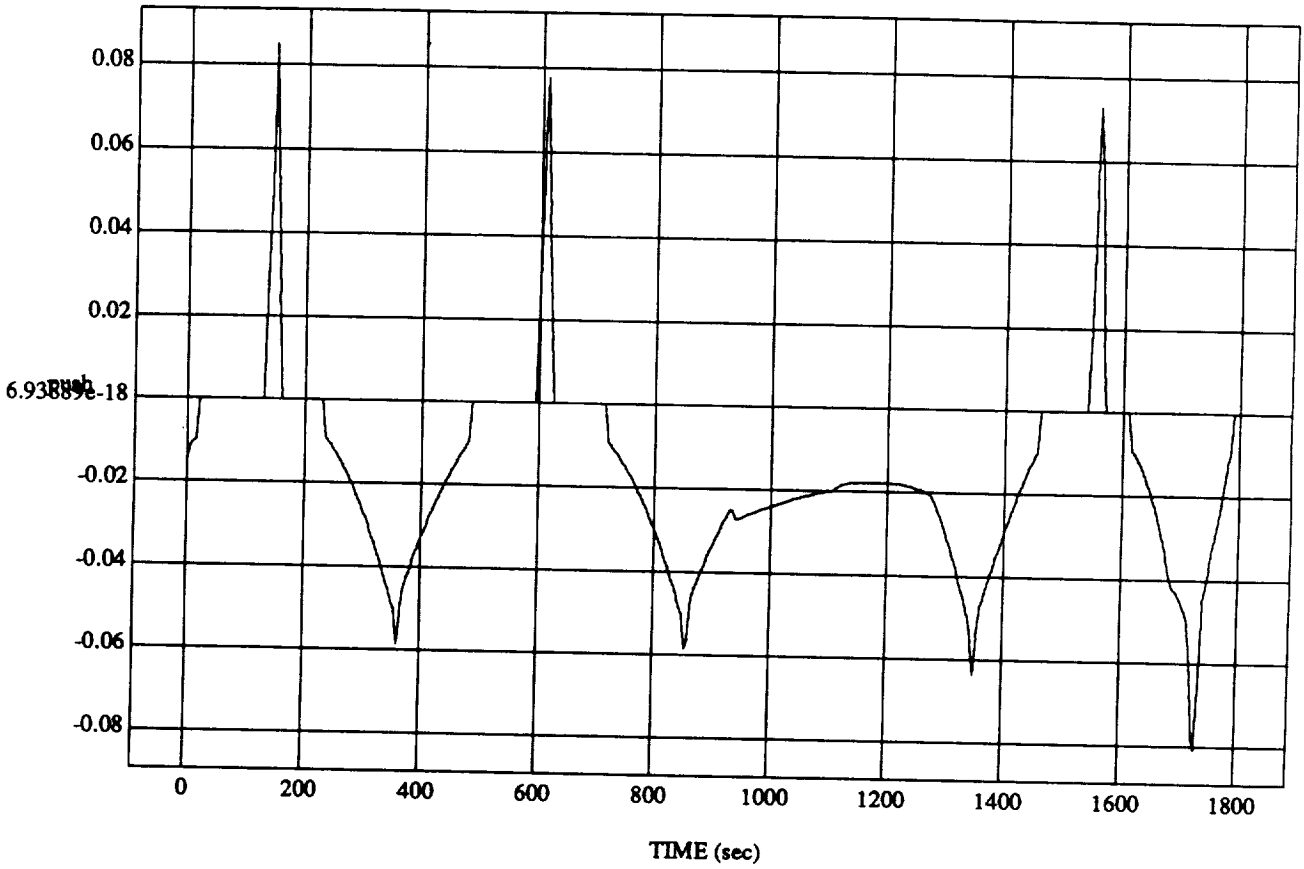


MODULE: ORBITER.lm_elev

DATA SAMPLING FREQUENCY: 0.200 Hz

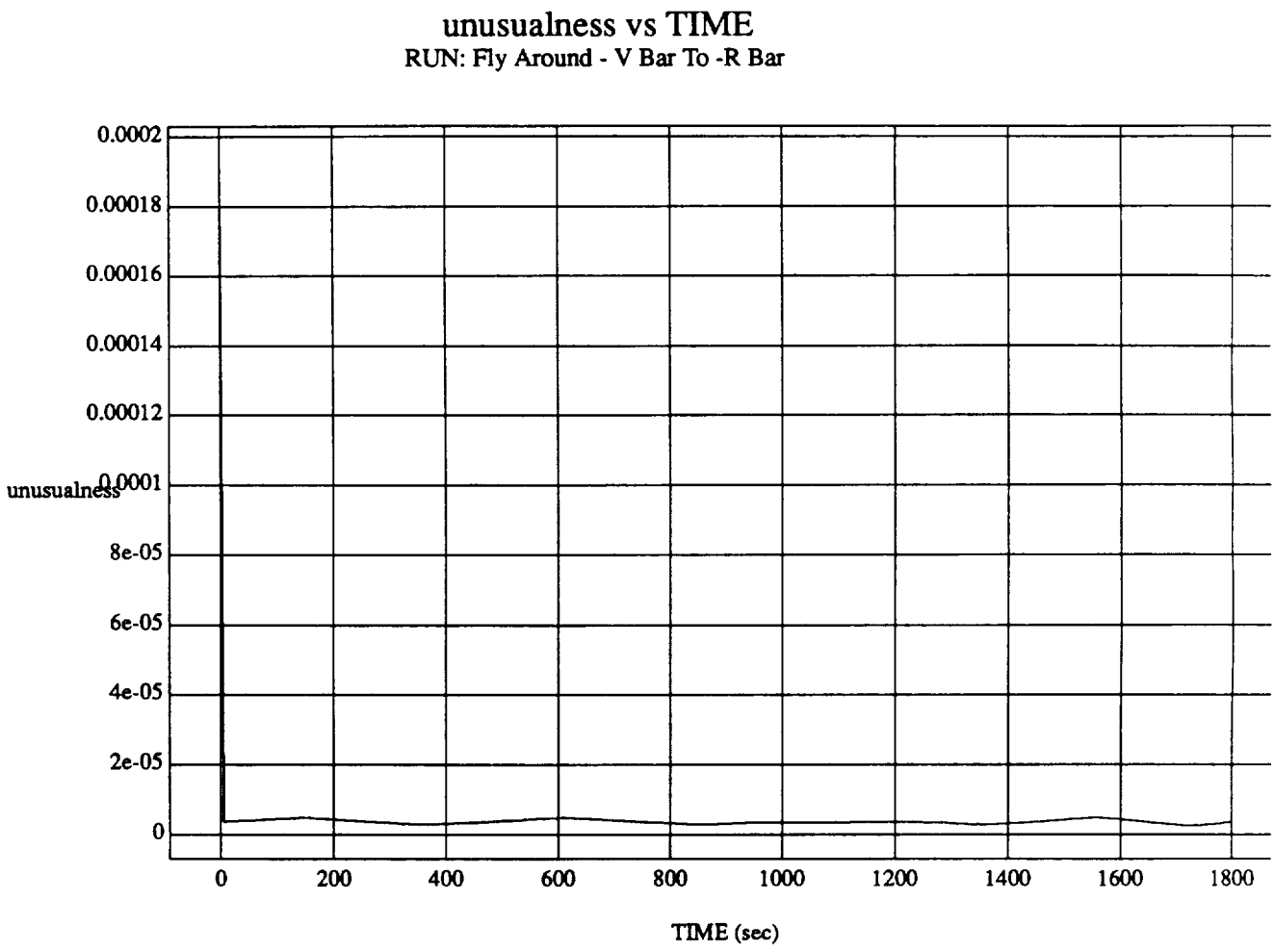
push vs TIME

RUN: Fly Around - V Bar To -R Bar

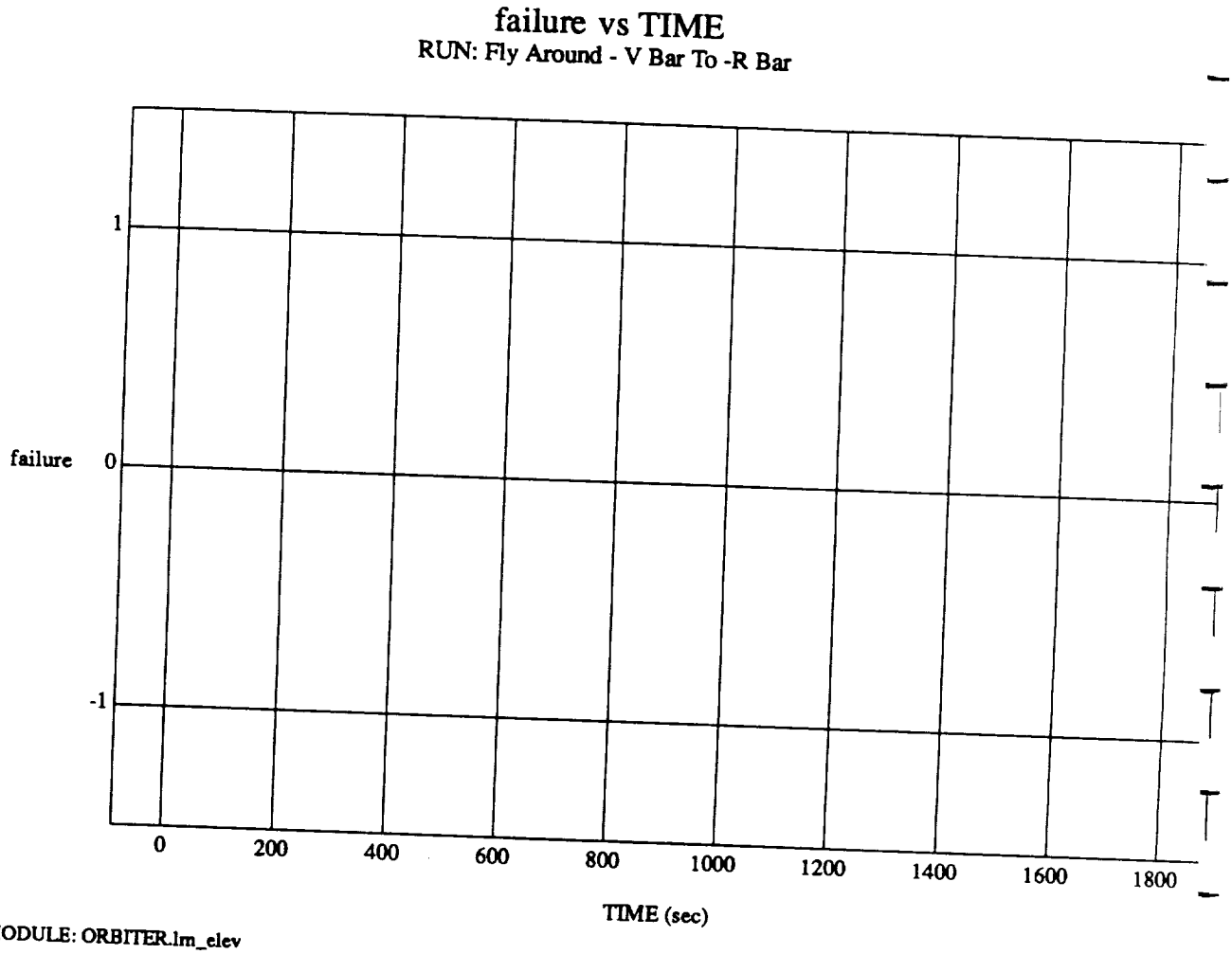


MODULE: ORBITER.lm_elev

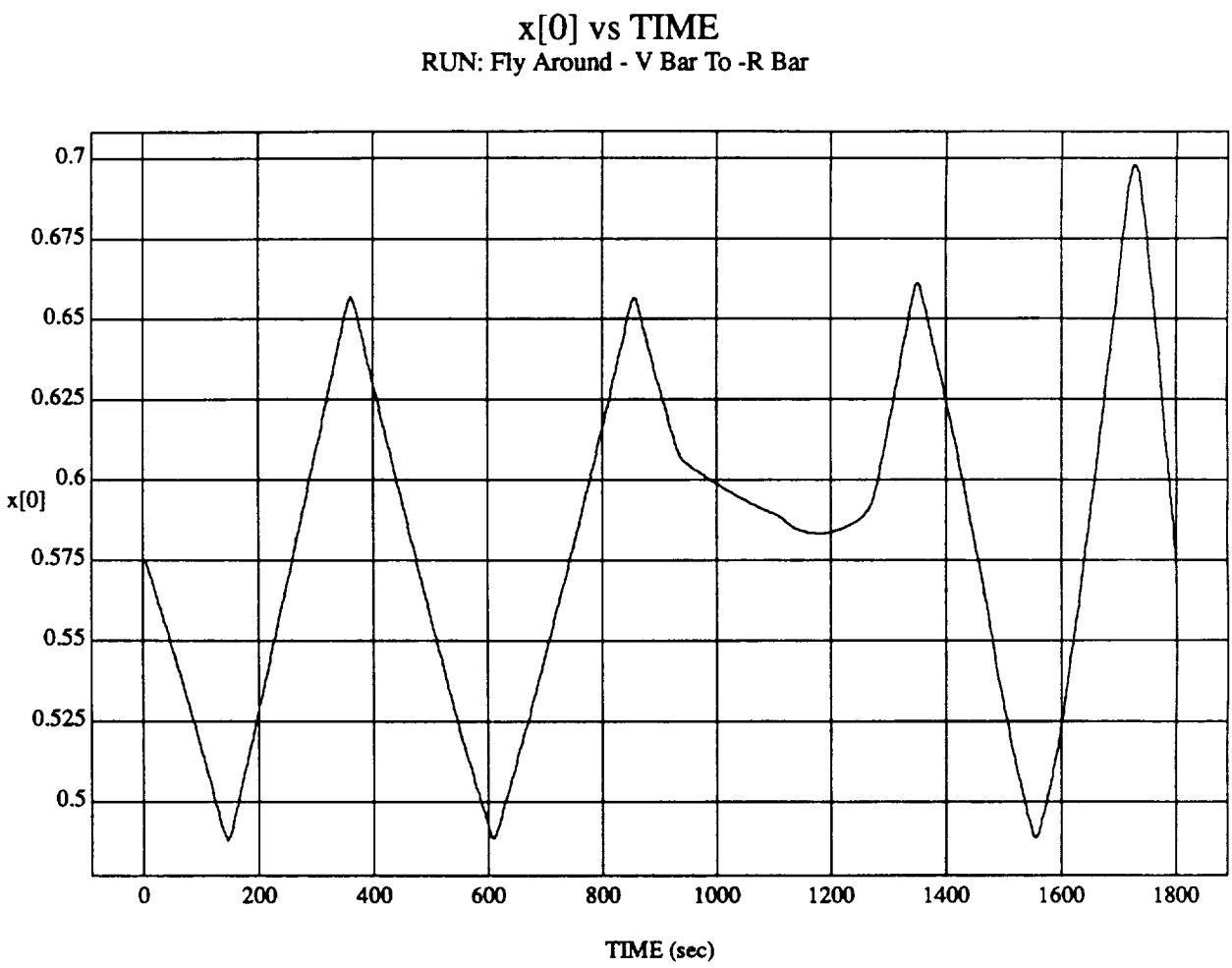
DATA SAMPLING FREQUENCY: 0.200 Hz



MODULE: ORBITER.lm_elev
DATA SAMPLING FREQUENCY: 0.200 Hz

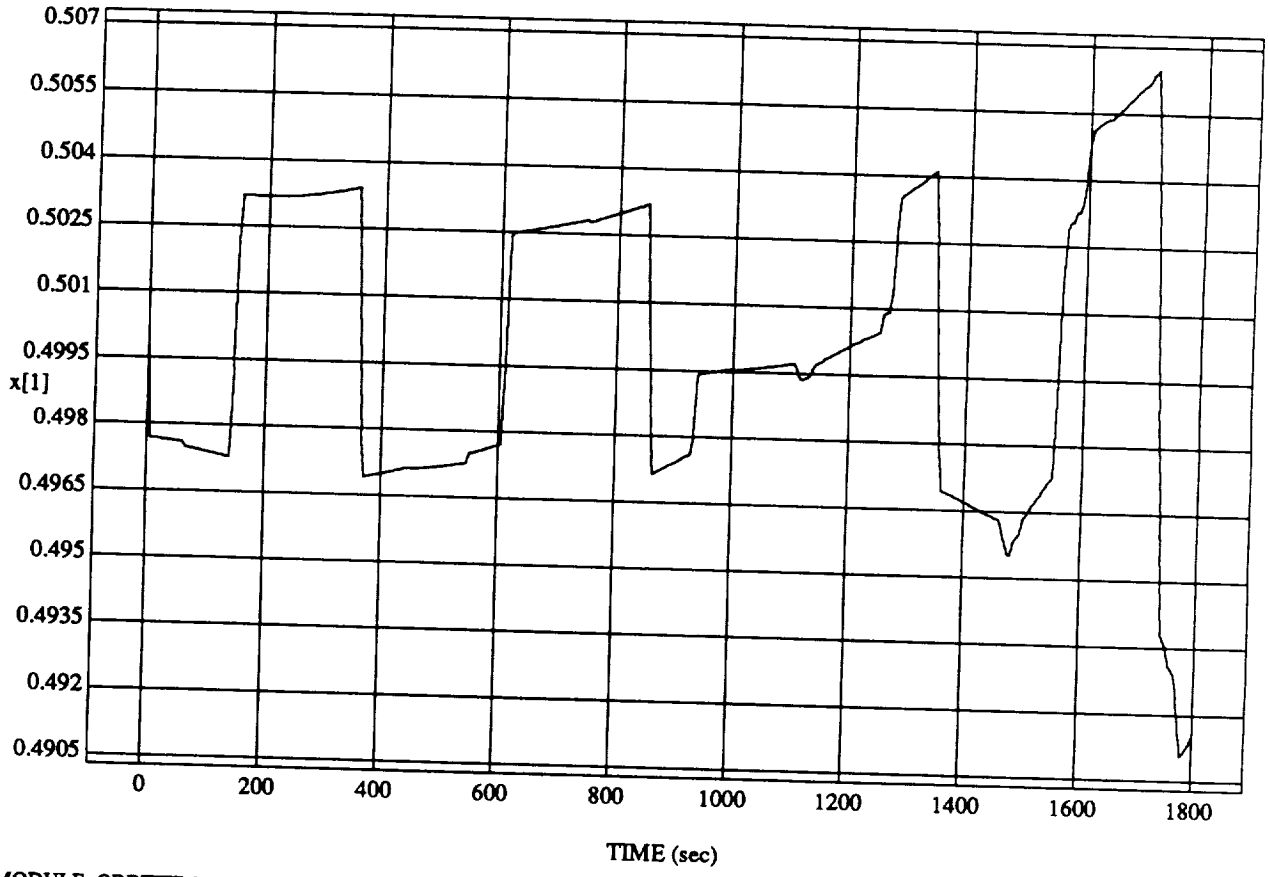


MODULE: ORBITER.lm_elev
DATA SAMPLING FREQUENCY: 0.200 Hz

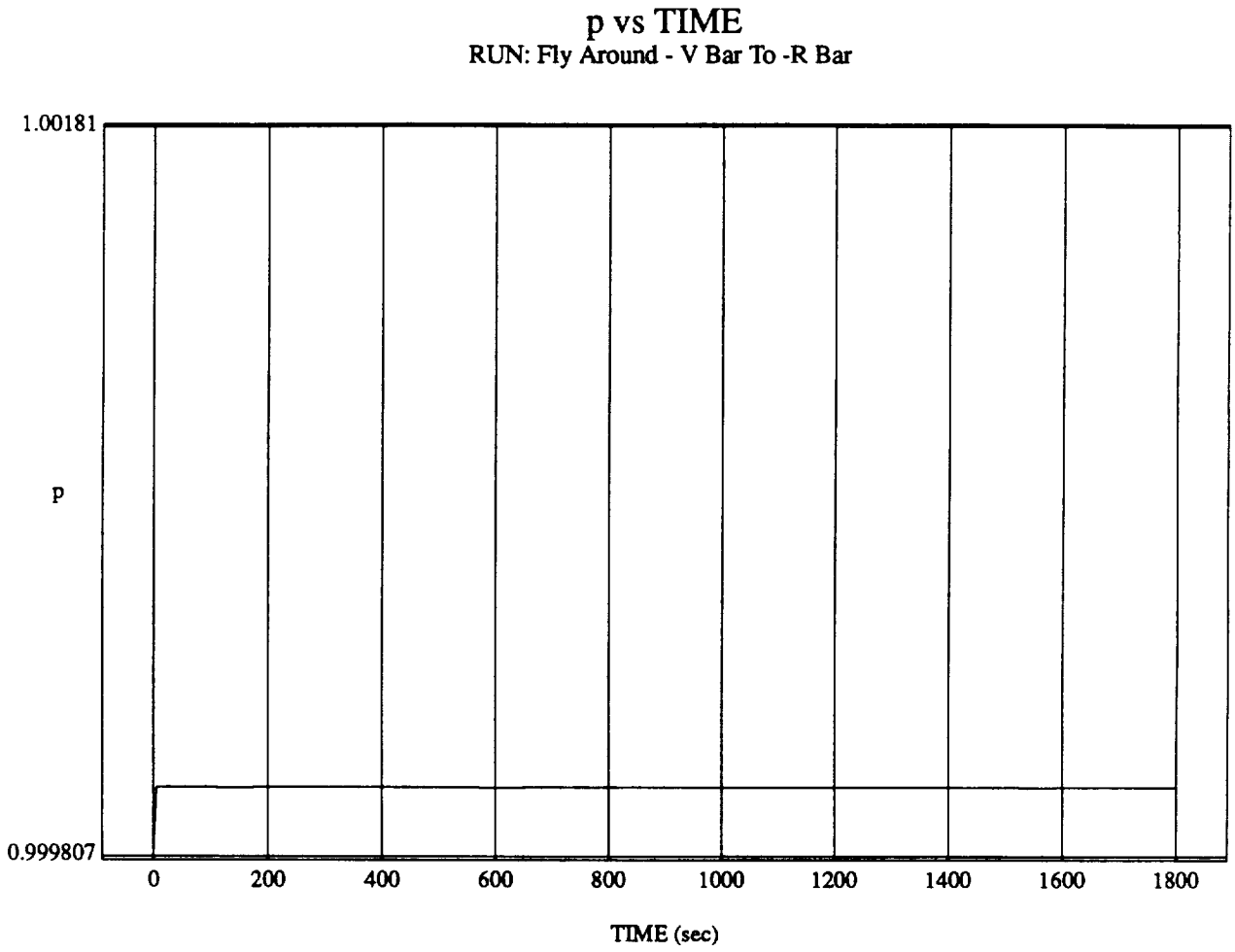


MODULE: ORBITER.lm_elev
DATA SAMPLING FREQUENCY: 0.200 Hz

x[1] vs TIME
RUN: Fly Around - V Bar To -R Bar



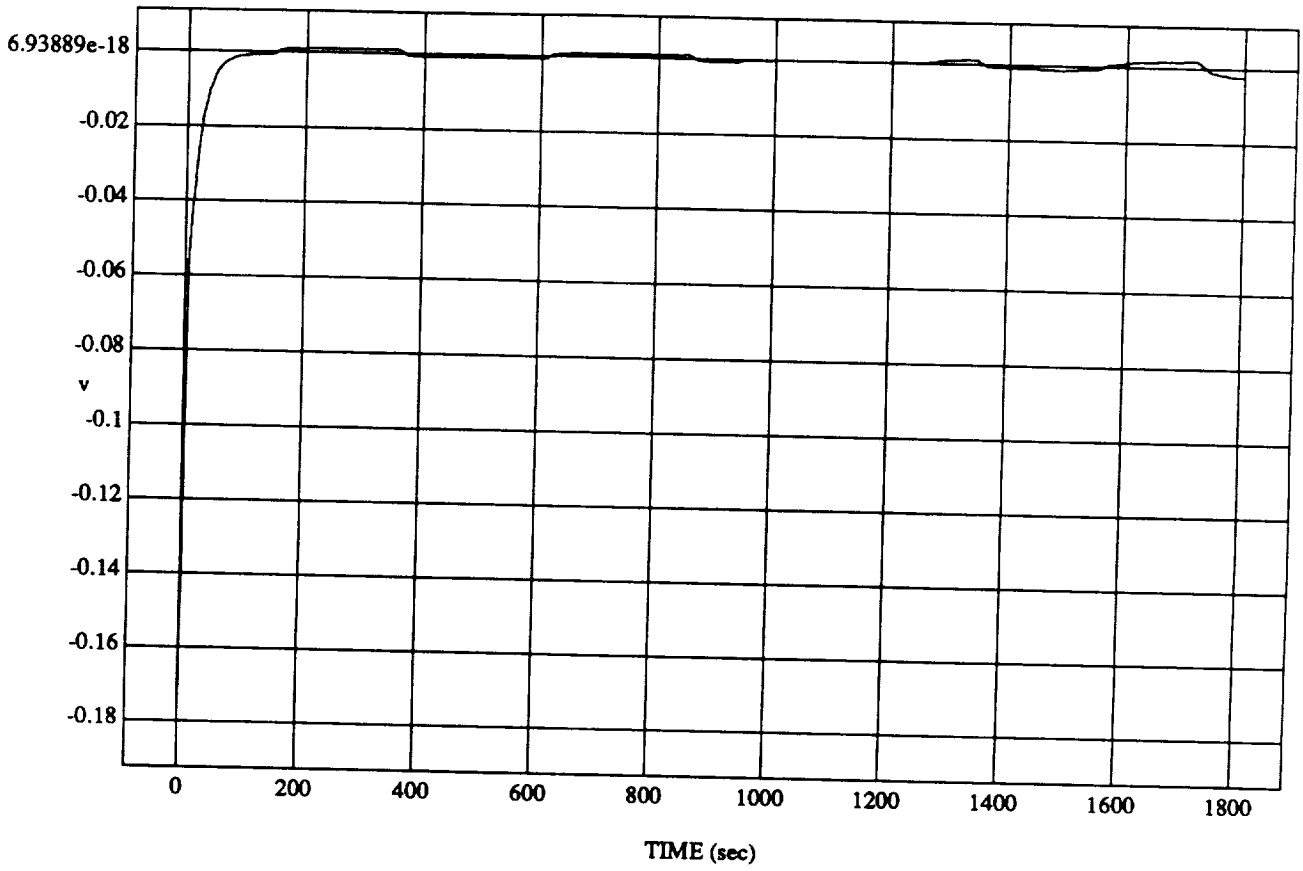
MODULE: ORBITER.lm_elev
DATA SAMPLING FREQUENCY: 0.200 Hz



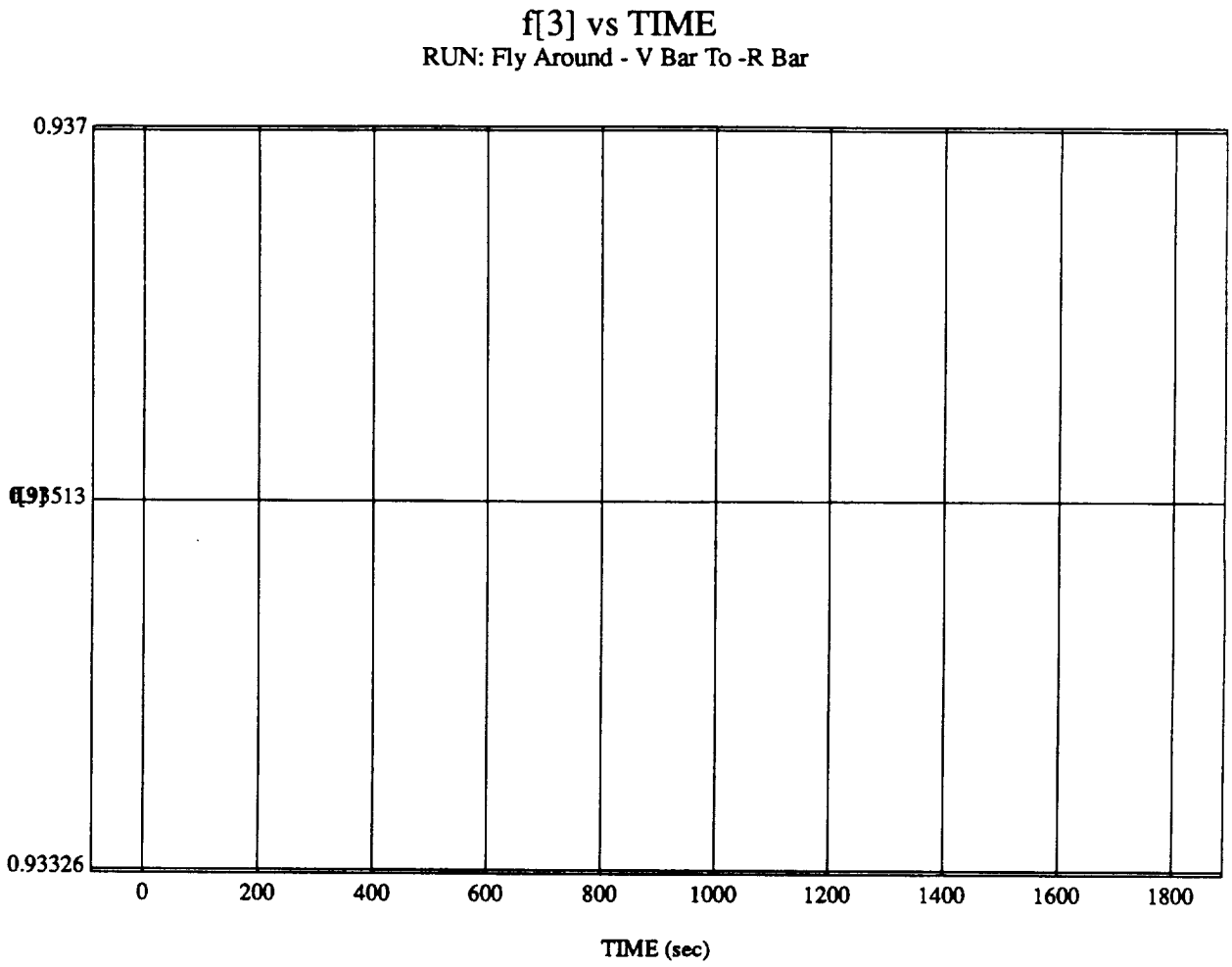
MODULE: ORBITER.lm_elev
DATA SAMPLING FREQUENCY: 0.200 Hz

v vs TIME

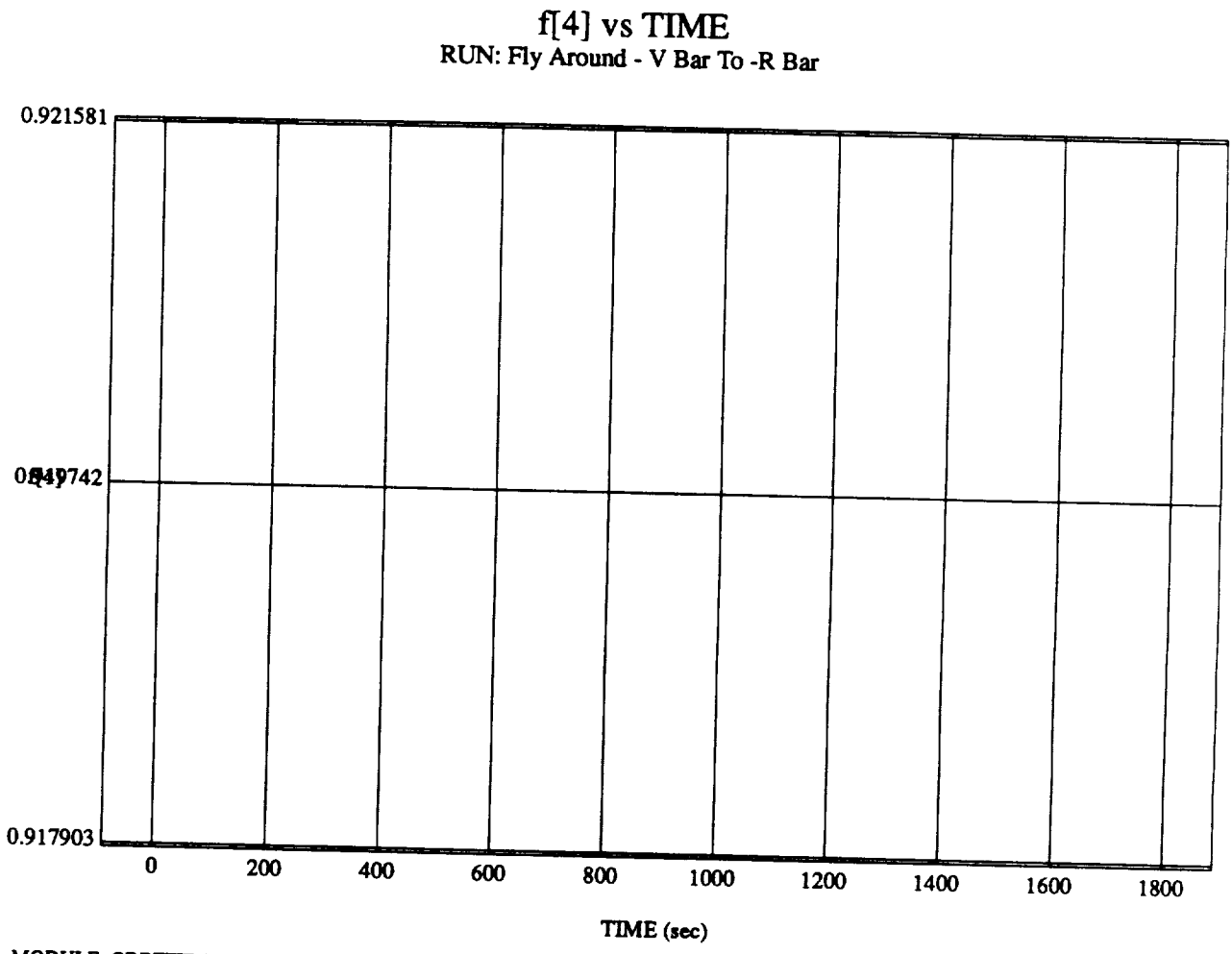
RUN: Fly Around - V Bar To -R Bar



MODULE: ORBITER.lm_elev
DATA SAMPLING FREQUENCY: 0.200 Hz



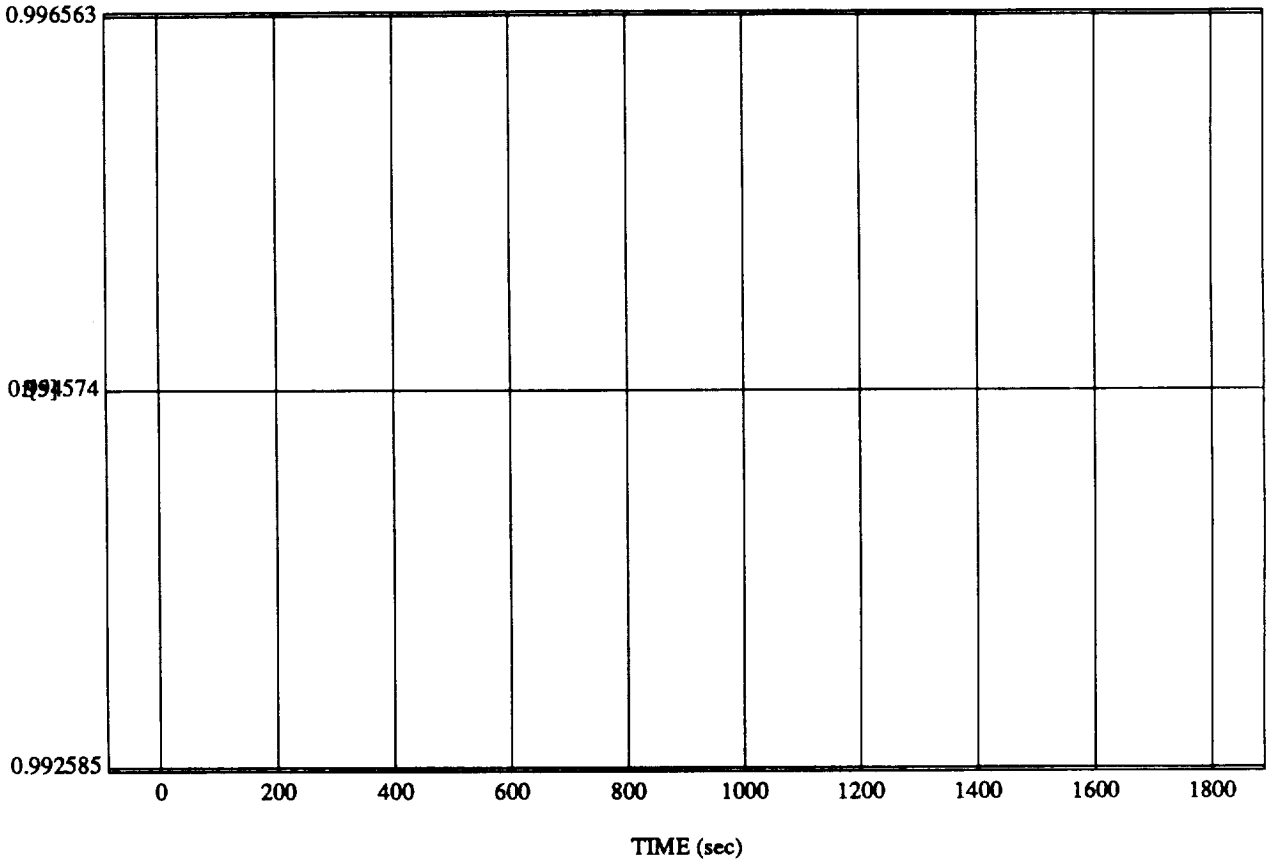
MODULE: ORBITER.lm_elev
DATA SAMPLING FREQUENCY: 0.200 Hz



MODULE: ORBITER.lm_elev
DATA SAMPLING FREQUENCY: 0.200 Hz

f[5] vs TIME

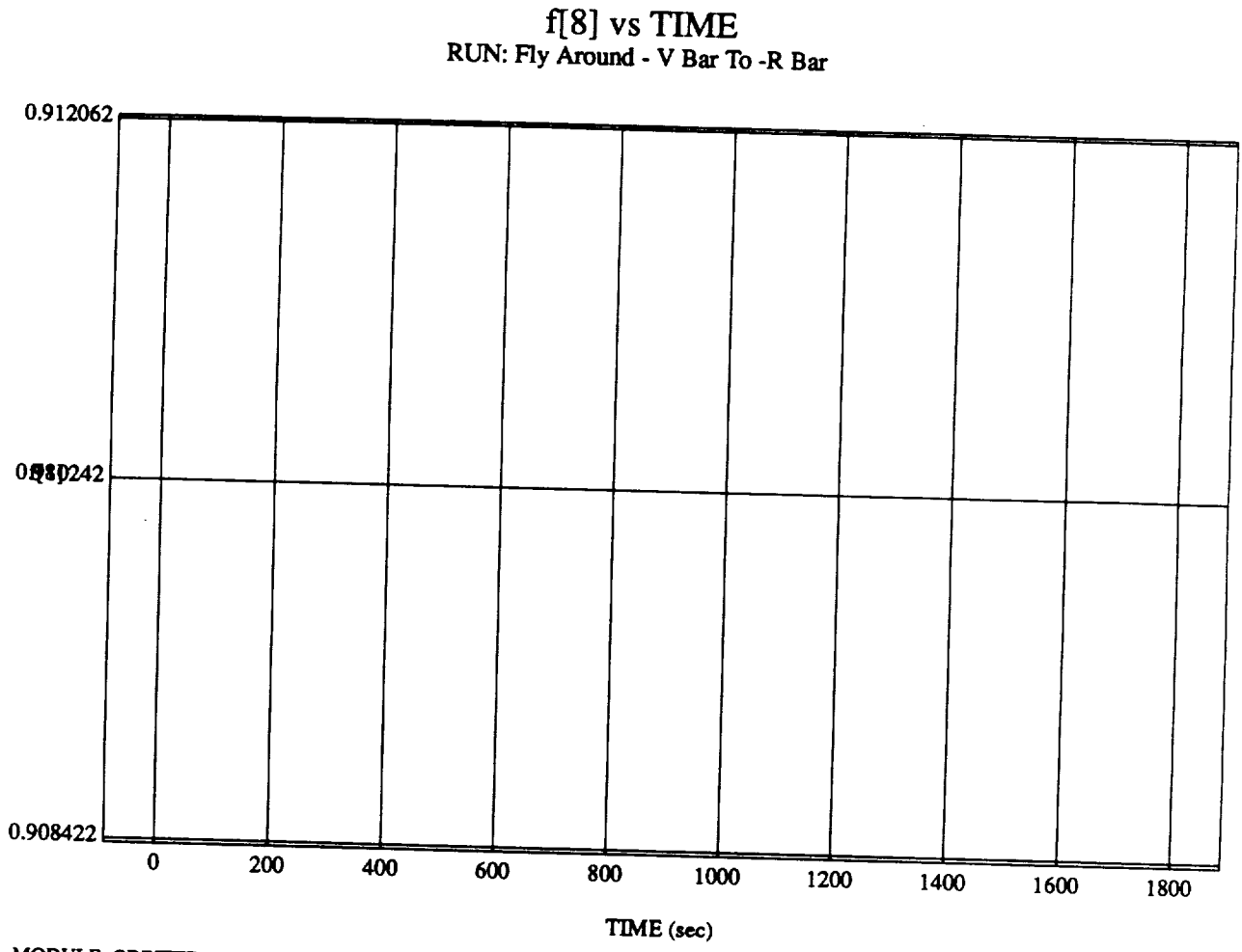
RUN: Fly Around - V Bar To -R Bar



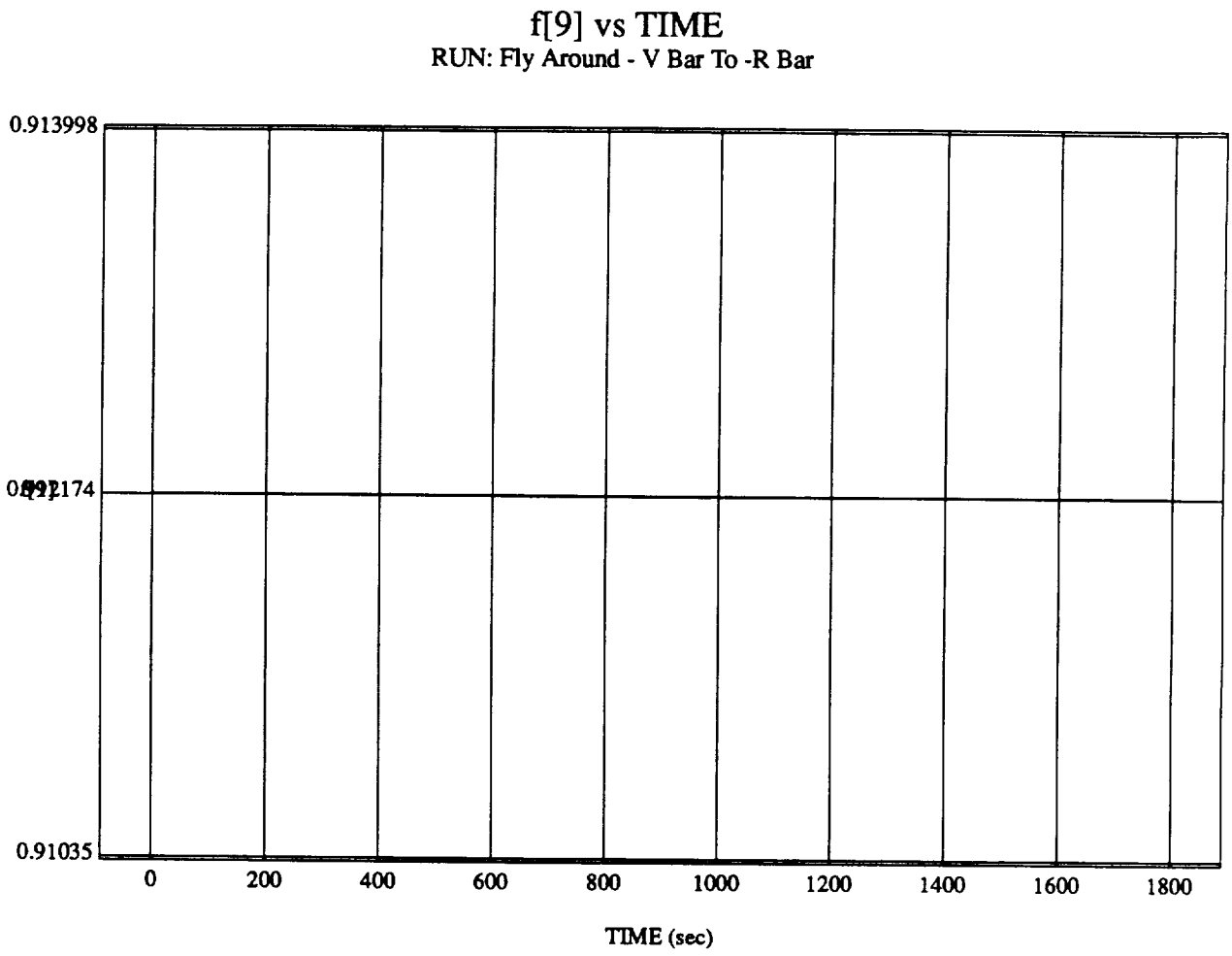
MODULE: ORBITER.lm_elev

DATA SAMPLING FREQUENCY: 0.200 Hz

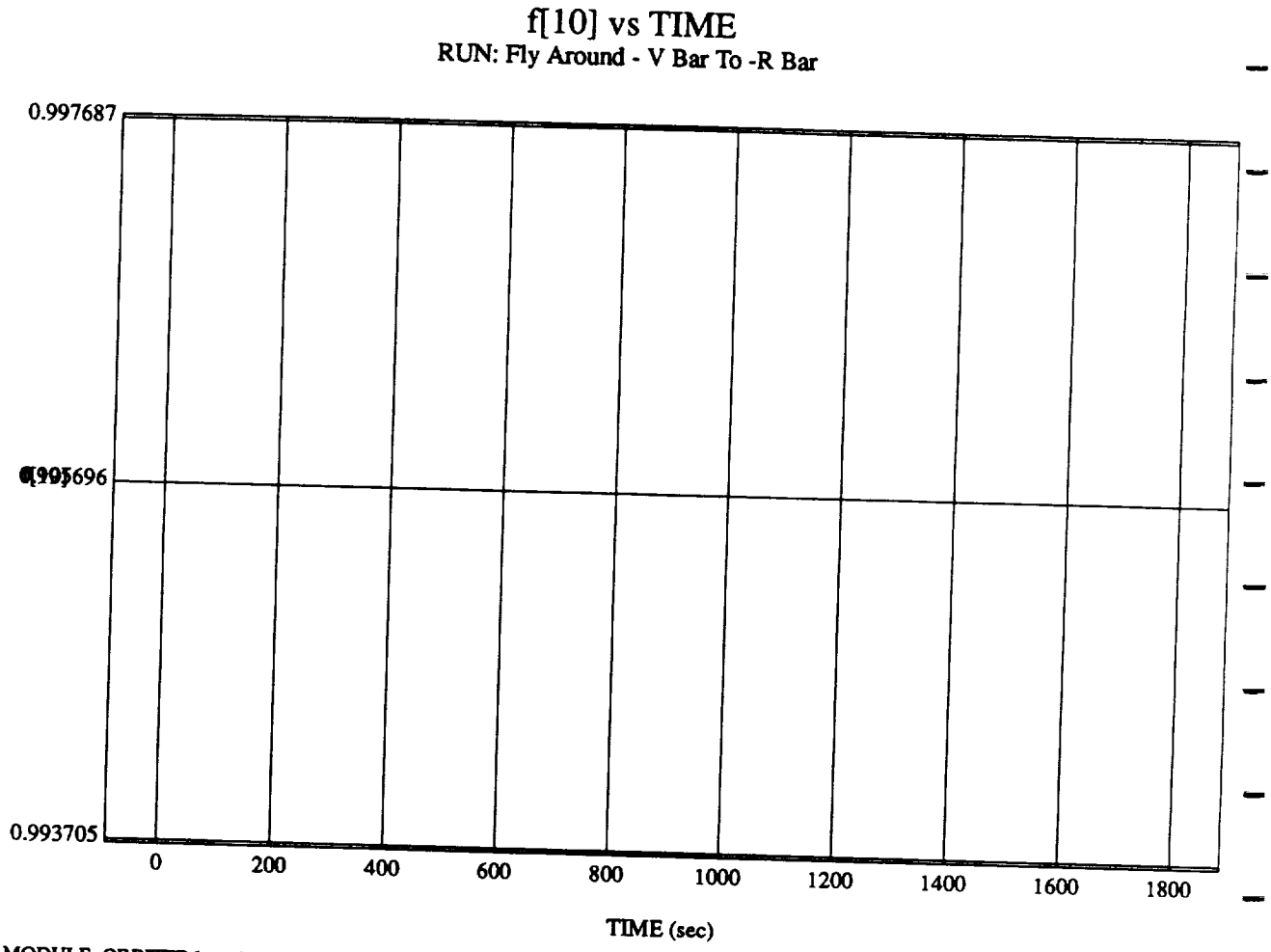
SIMULATION APPLICATION: ARIC Translational Controller Simulation



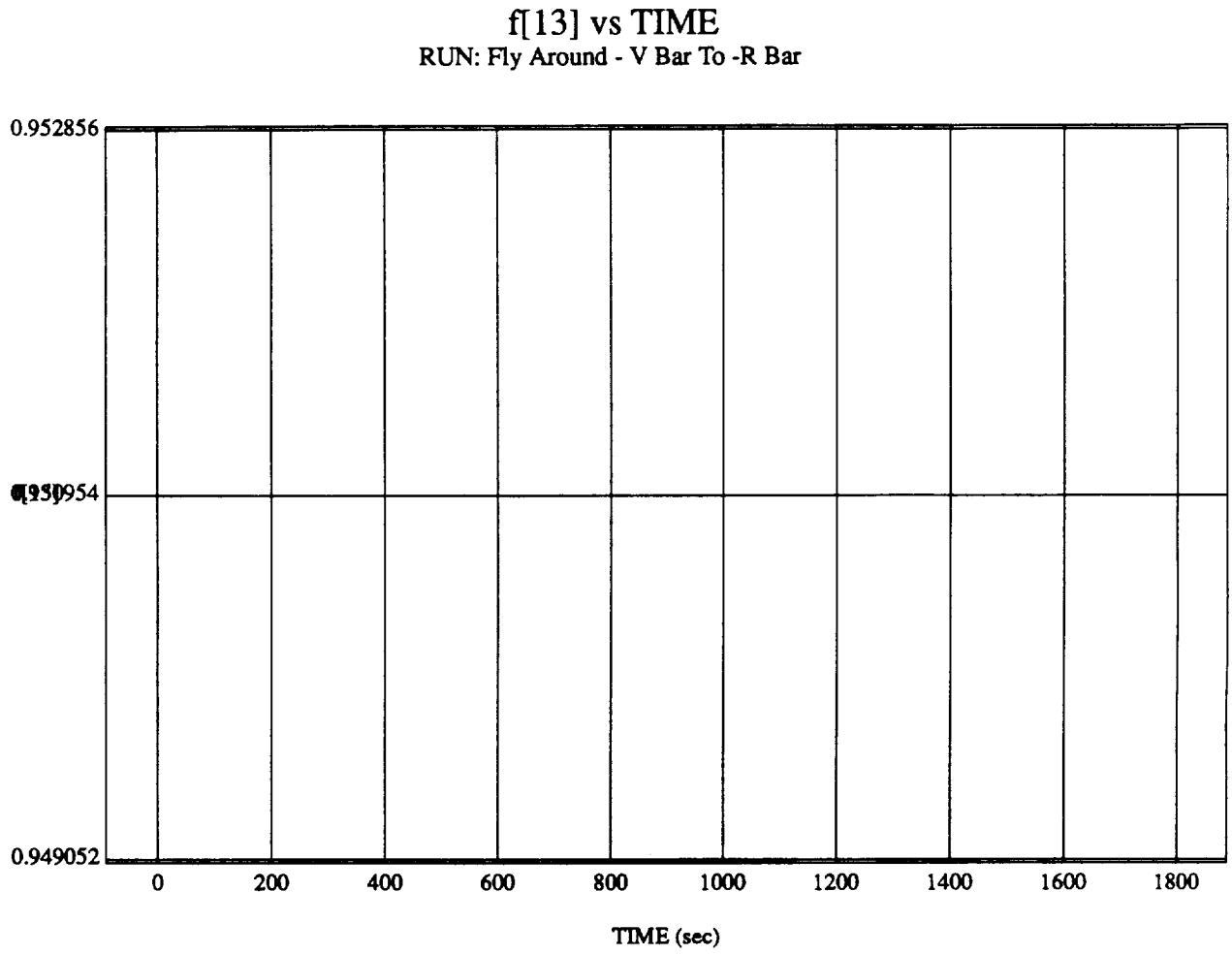
MODULE: ORBITER.lm_elev
DATA SAMPLING FREQUENCY: 0.200 Hz



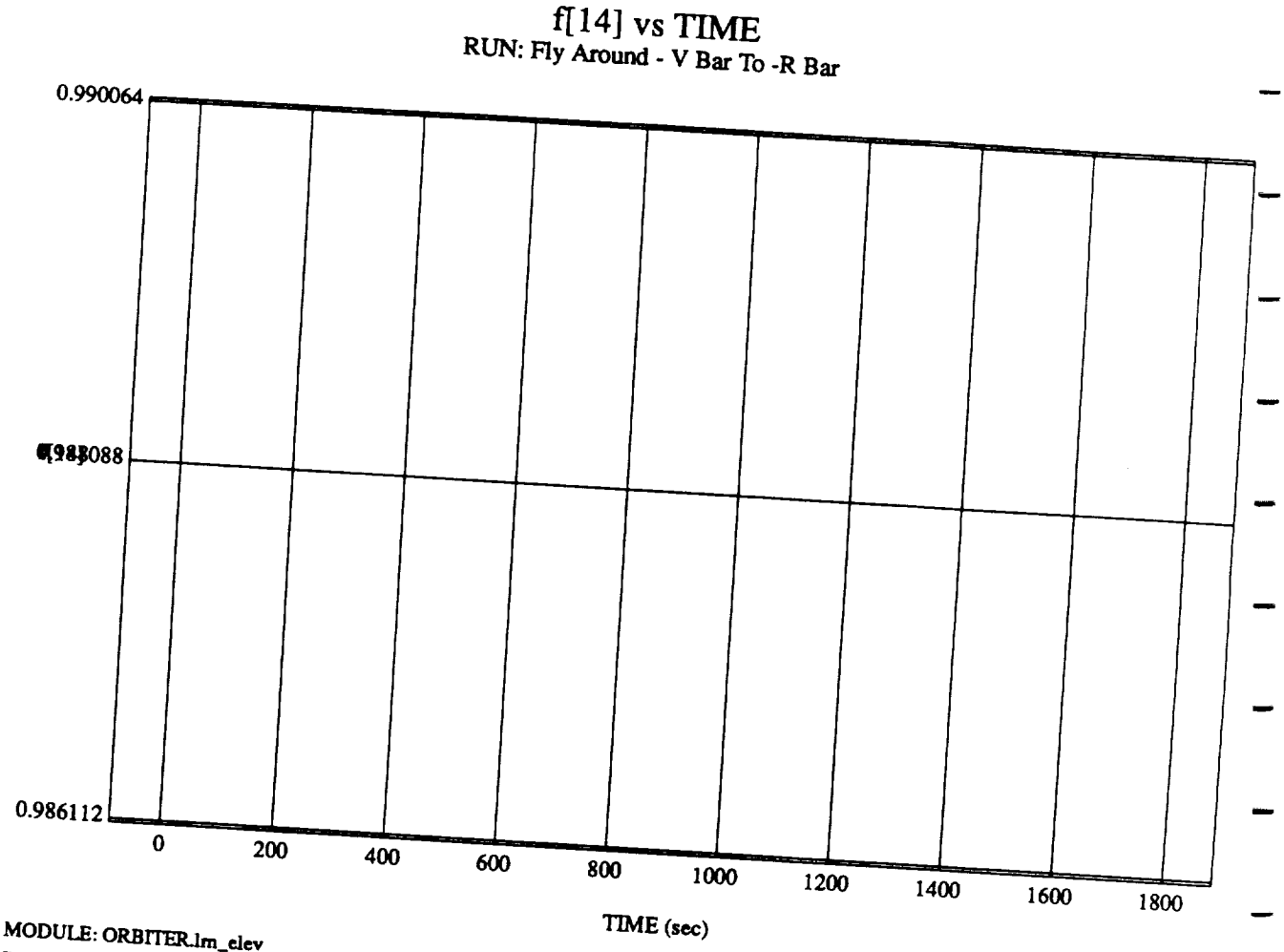
MODULE: ORBITER.lm_elev
DATA SAMPLING FREQUENCY: 0.200 Hz



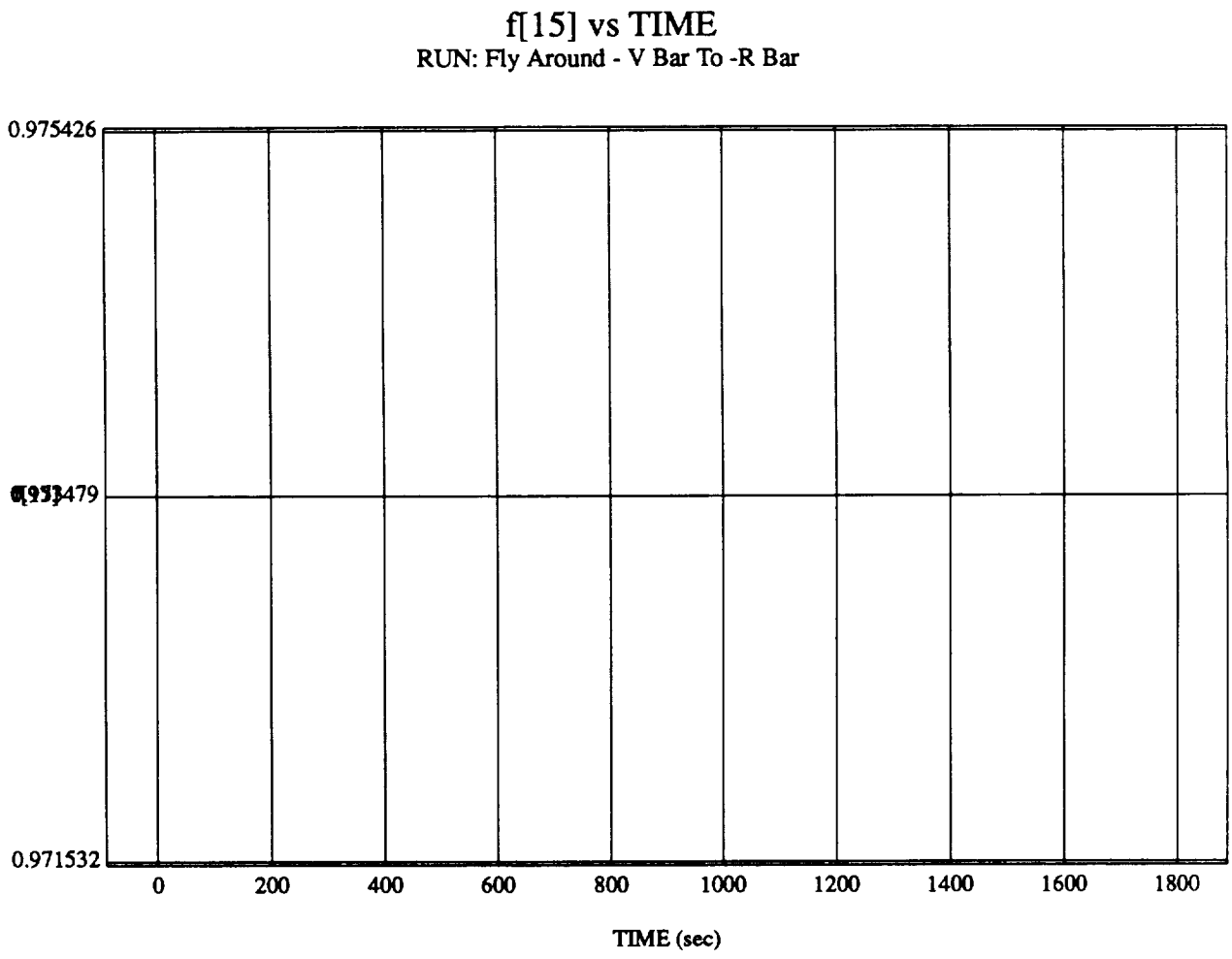
MODULE: ORBITER.lm_elev
DATA SAMPLING FREQUENCY: 0.200 Hz



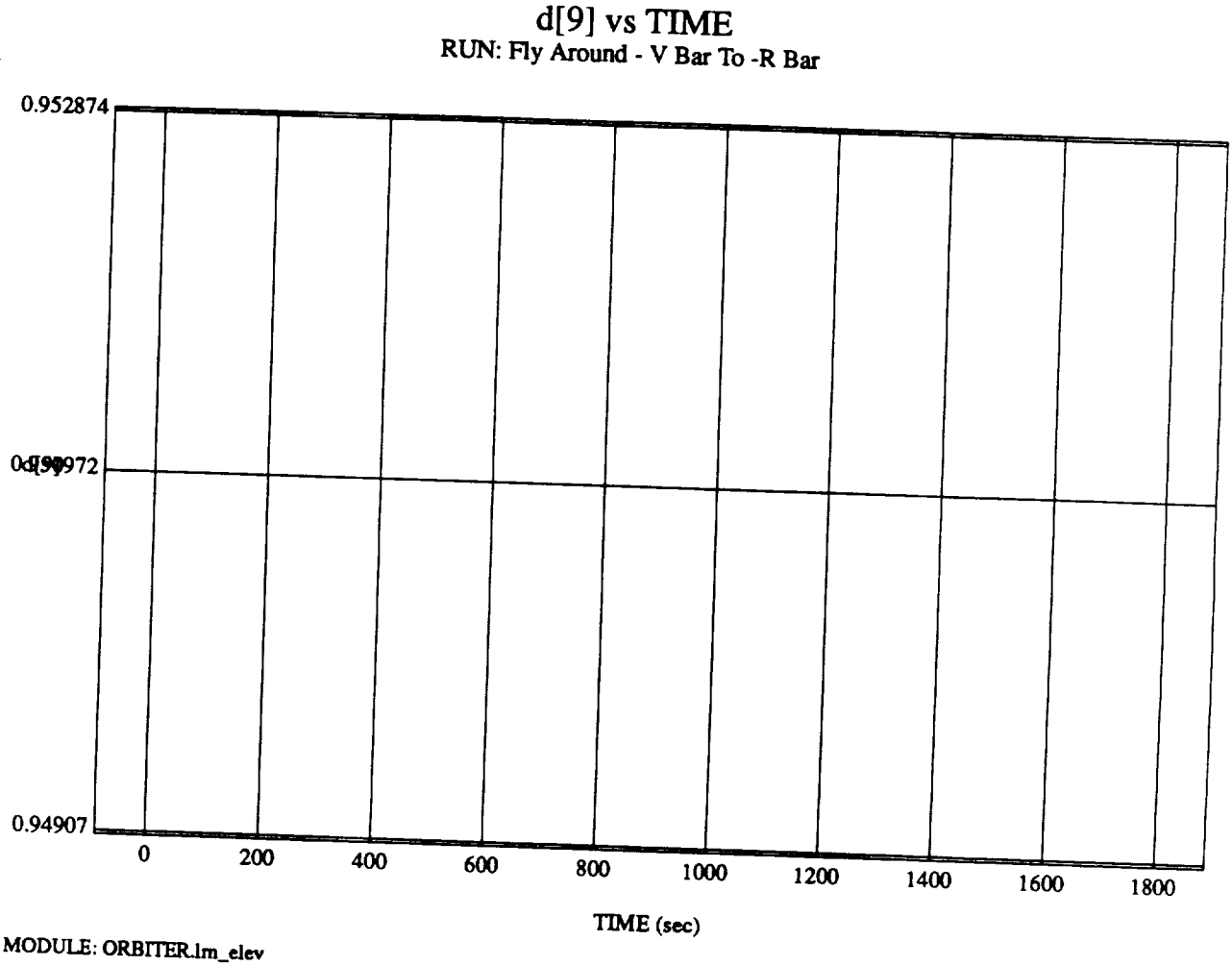
MODULE: ORBITER.lm_elev
DATA SAMPLING FREQUENCY: 0.200 Hz



MODULE: ORBITER.lm_elev
DATA SAMPLING FREQUENCY: 0.200 Hz

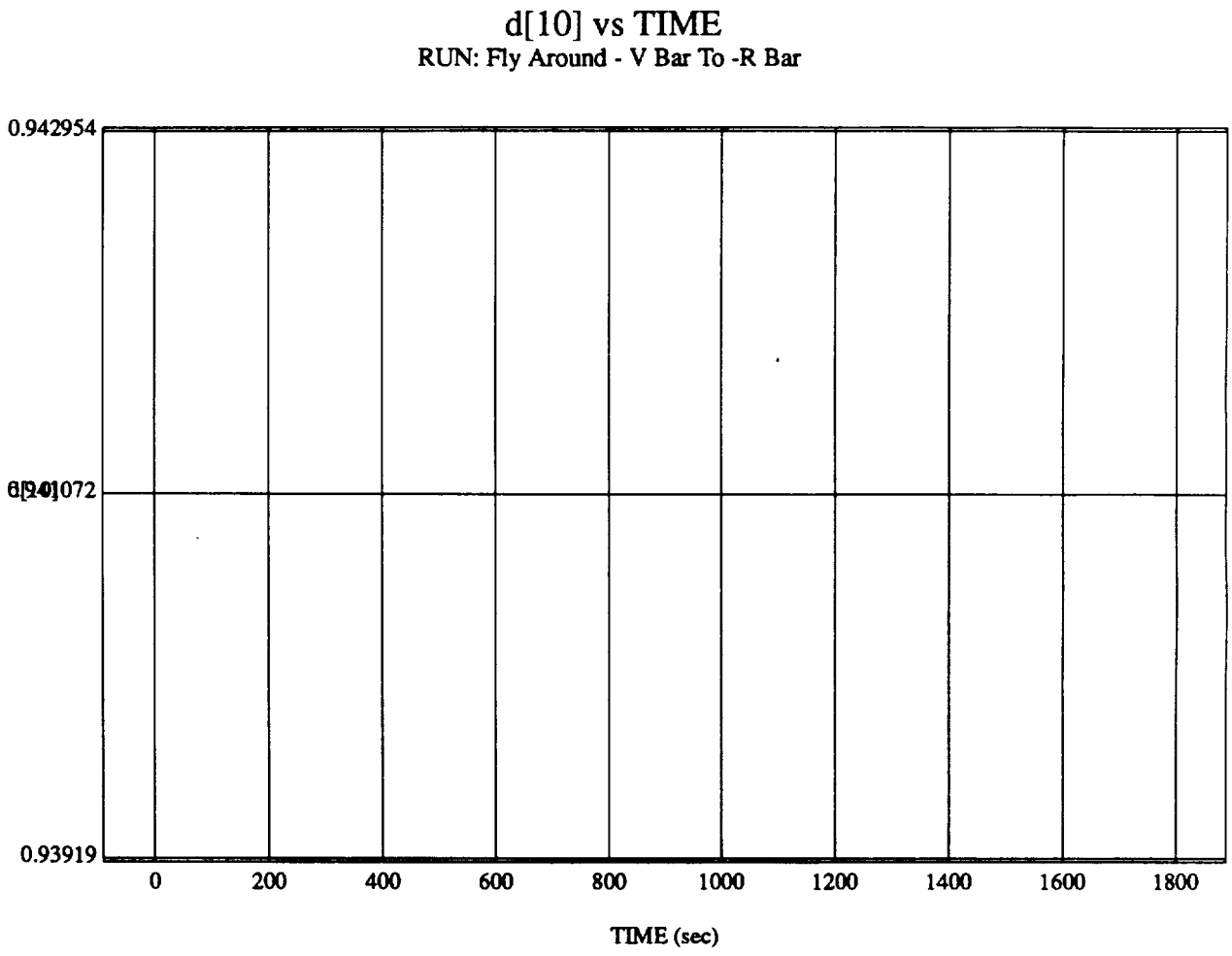


MODULE: ORBITER.lm_elev
DATA SAMPLING FREQUENCY: 0.200 Hz

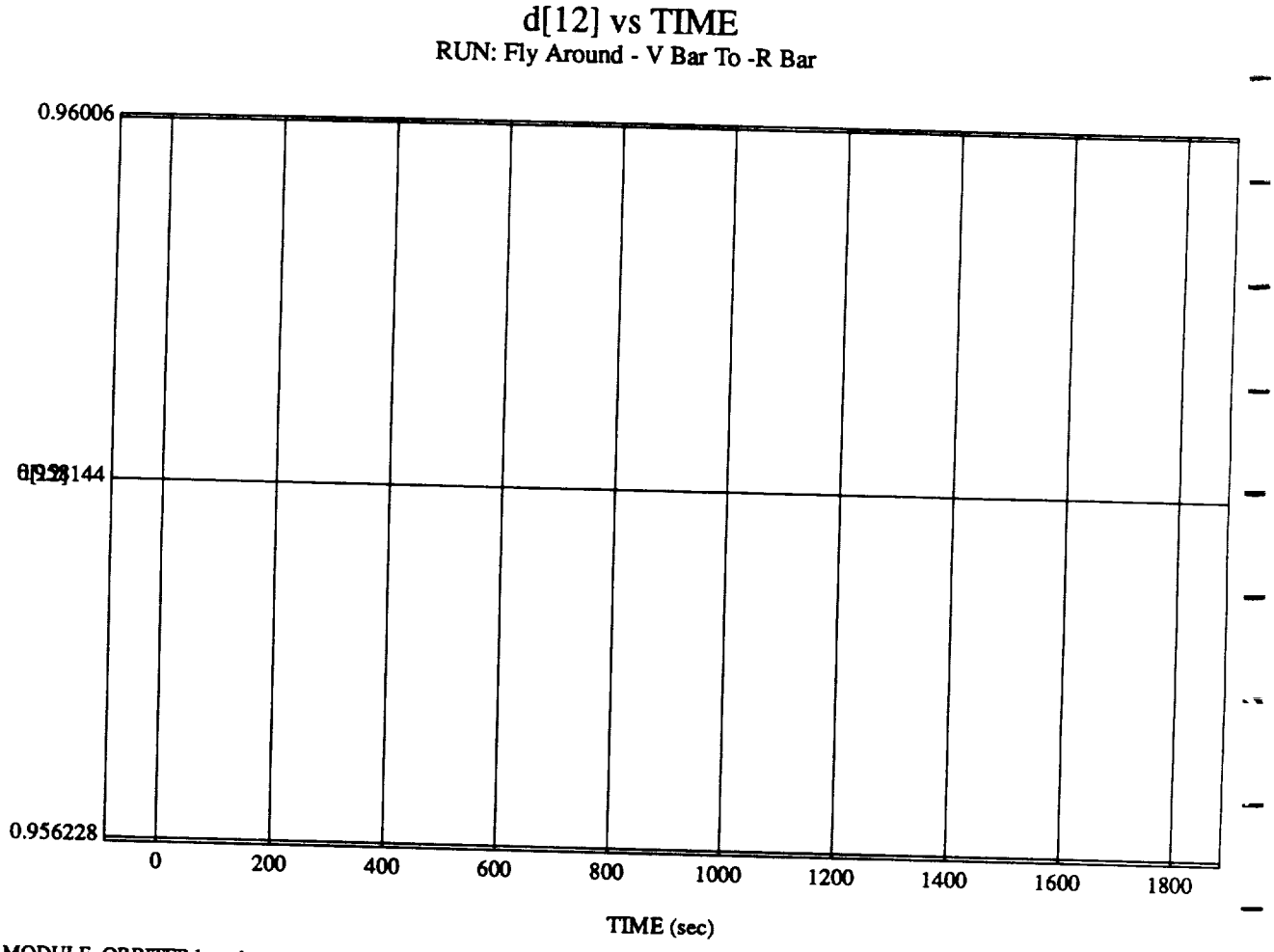


MODULE: ORBITER.lm_elev
DATA SAMPLING FREQUENCY: 0.200 Hz

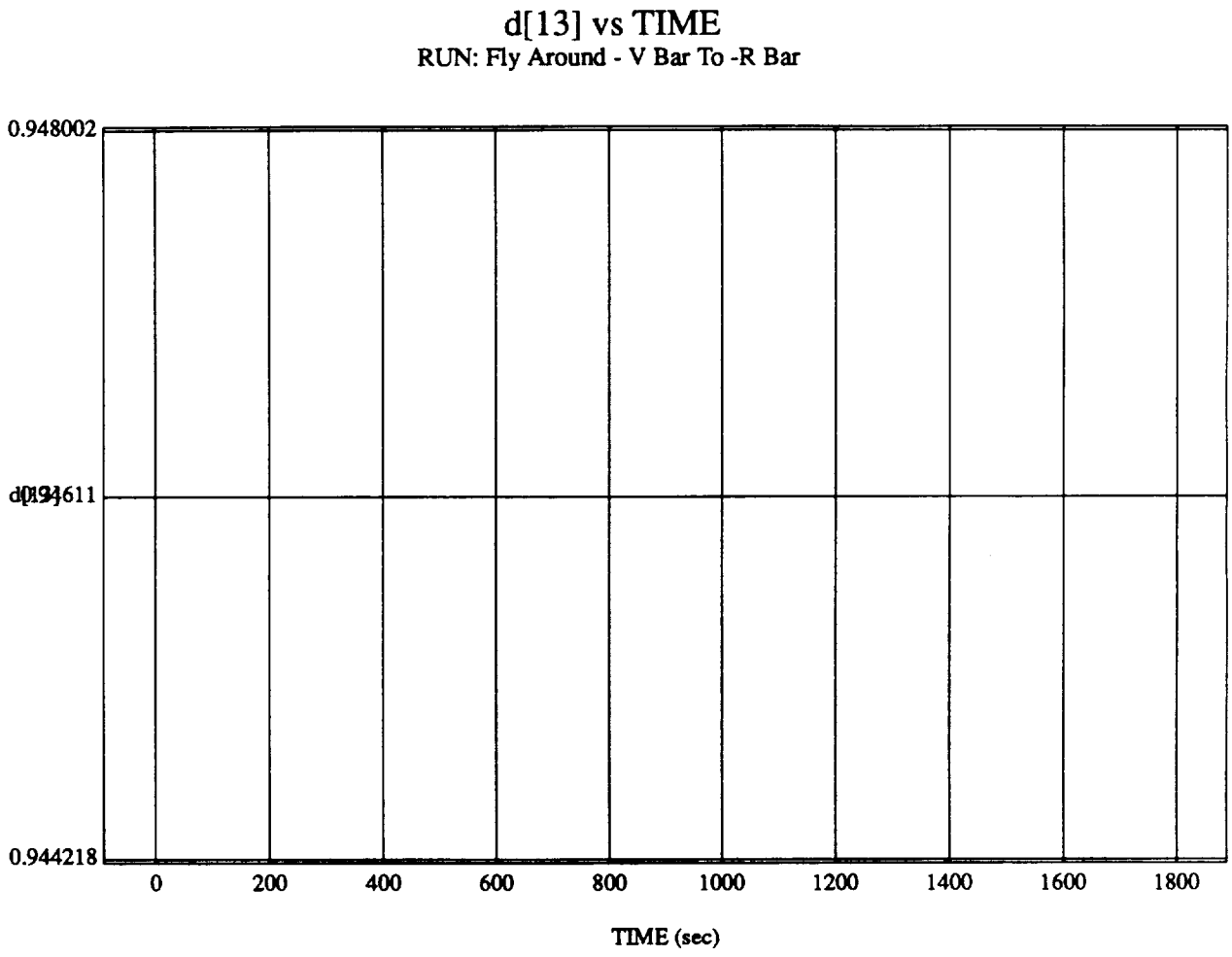
SIMULATION APPLICATION: ARIC Translational Controller Simulation



MODULE: ORBITER.lm_elev
DATA SAMPLING FREQUENCY: 0.200 Hz

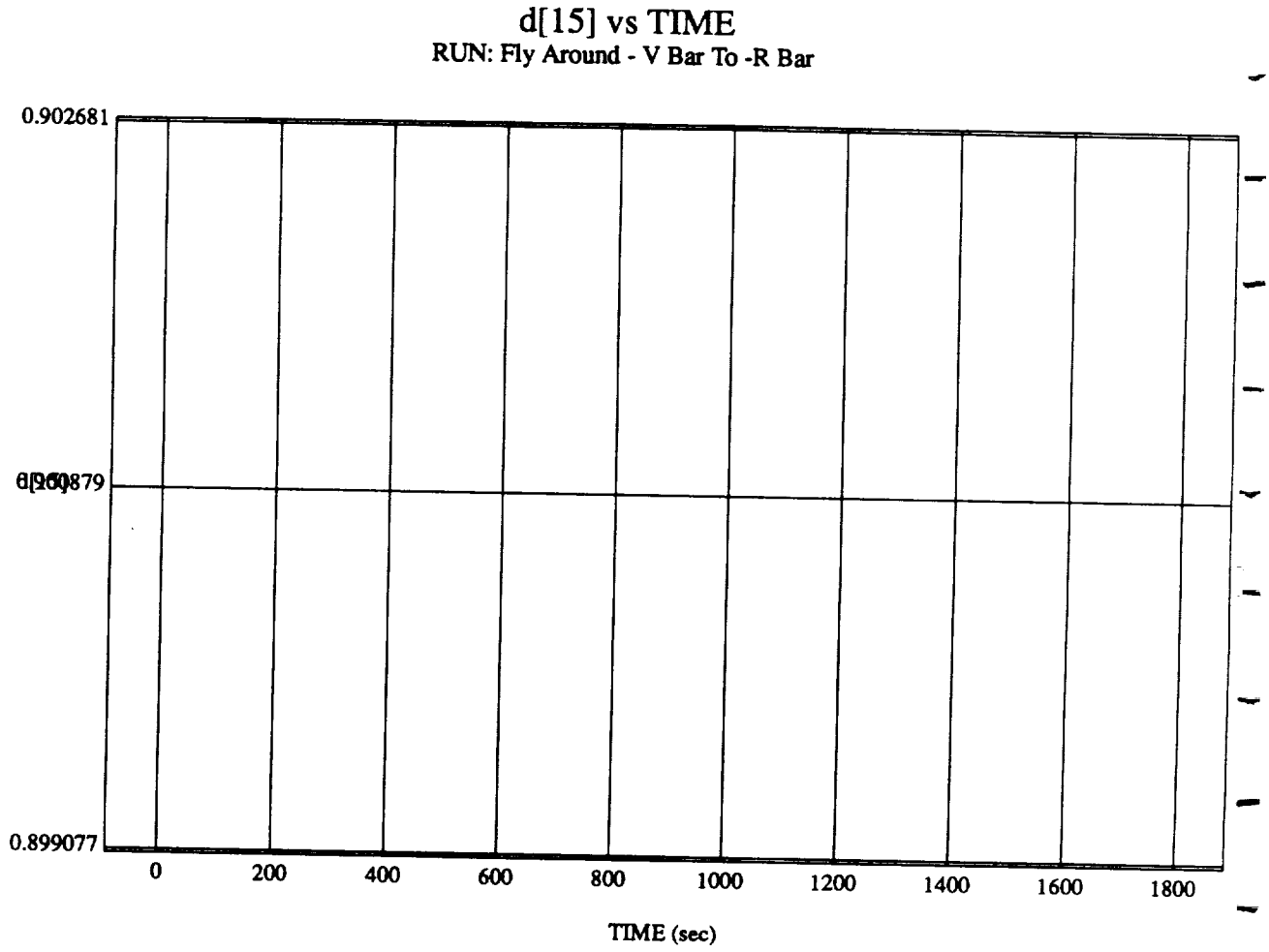


MODULE: ORBITER.lm_elev
DATA SAMPLING FREQUENCY: 0.200 Hz

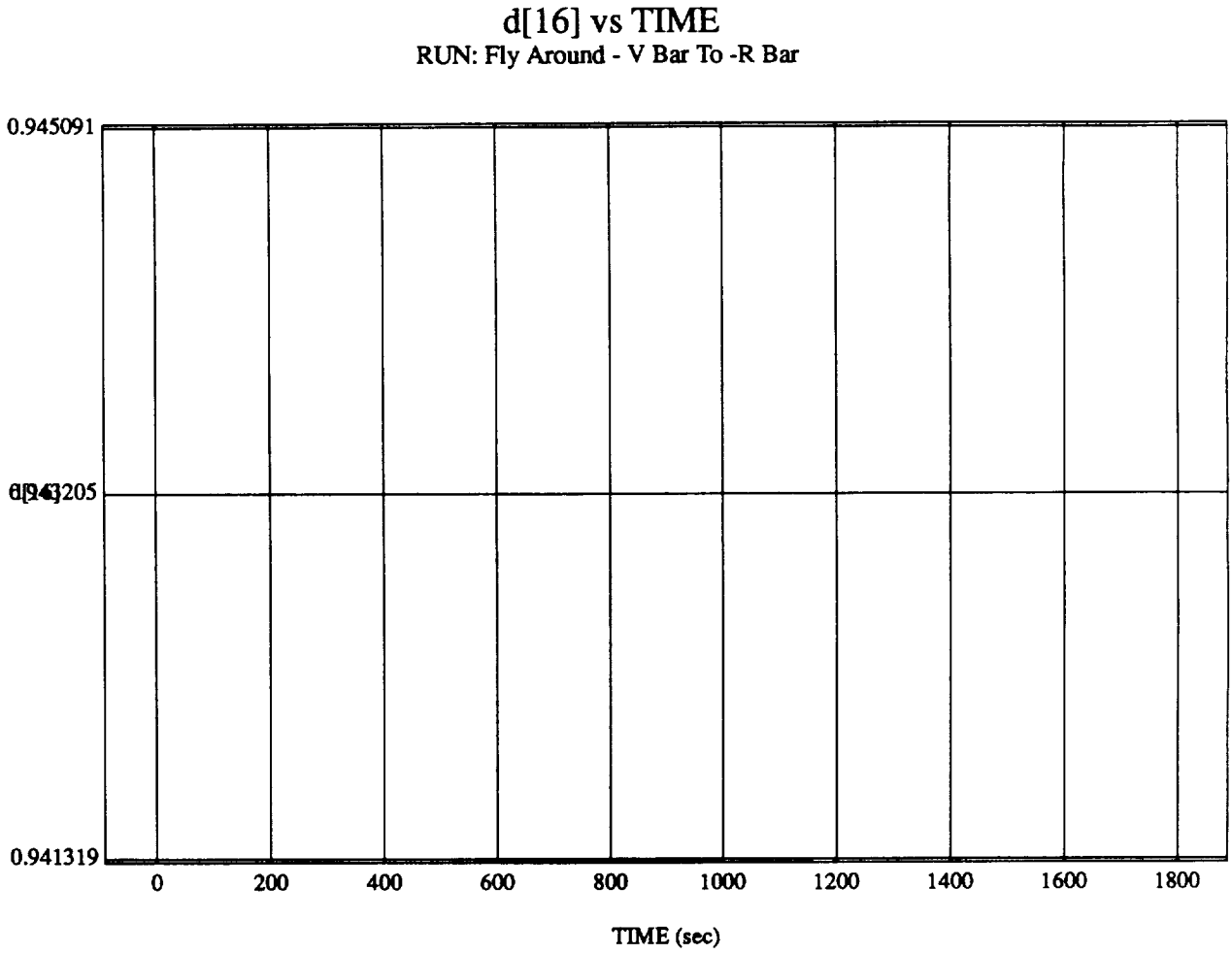


MODULE: ORBITER.lm_elev
DATA SAMPLING FREQUENCY: 0.200 Hz

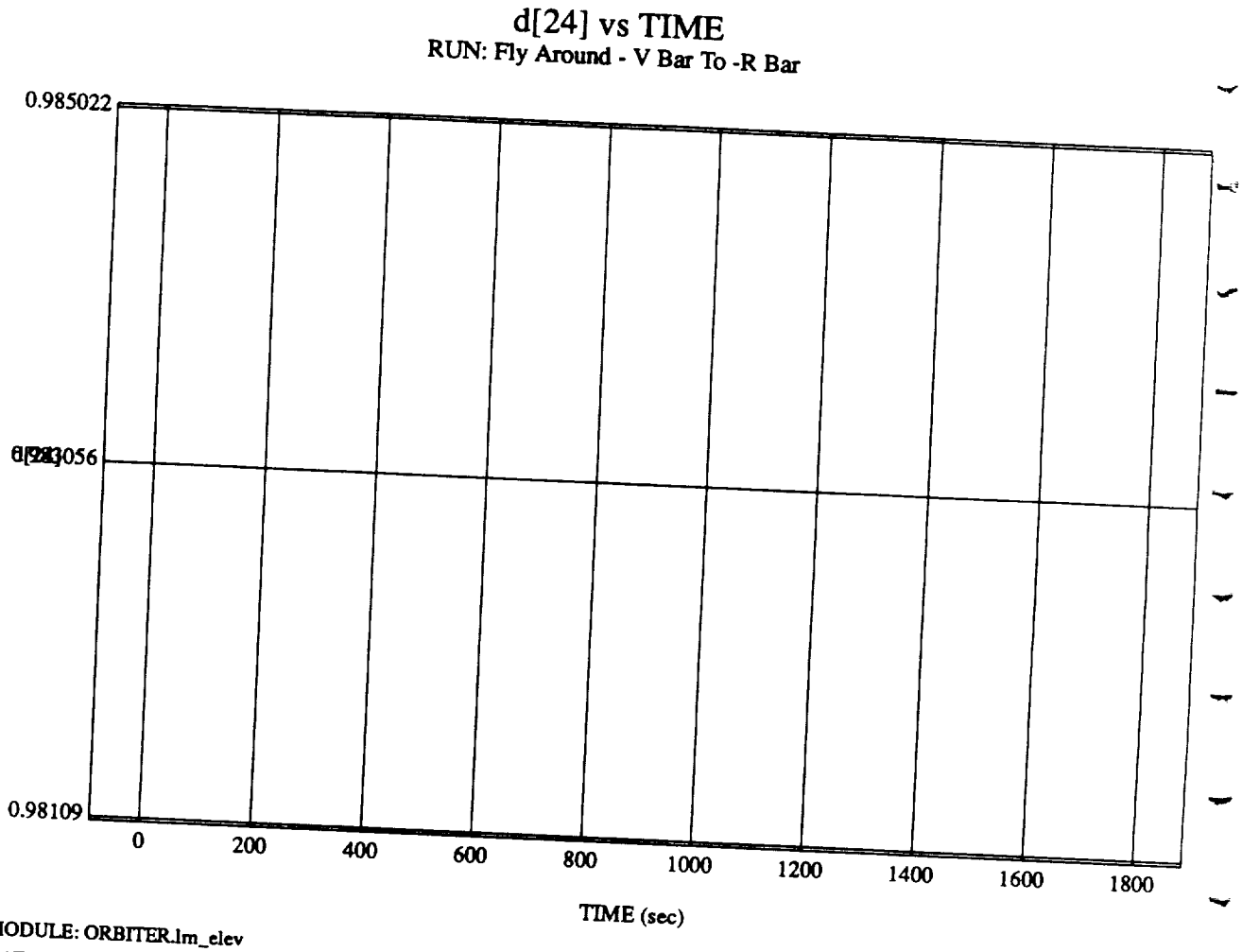
SIMULATION APPLICATION: ARIC Translational Controller Simulation



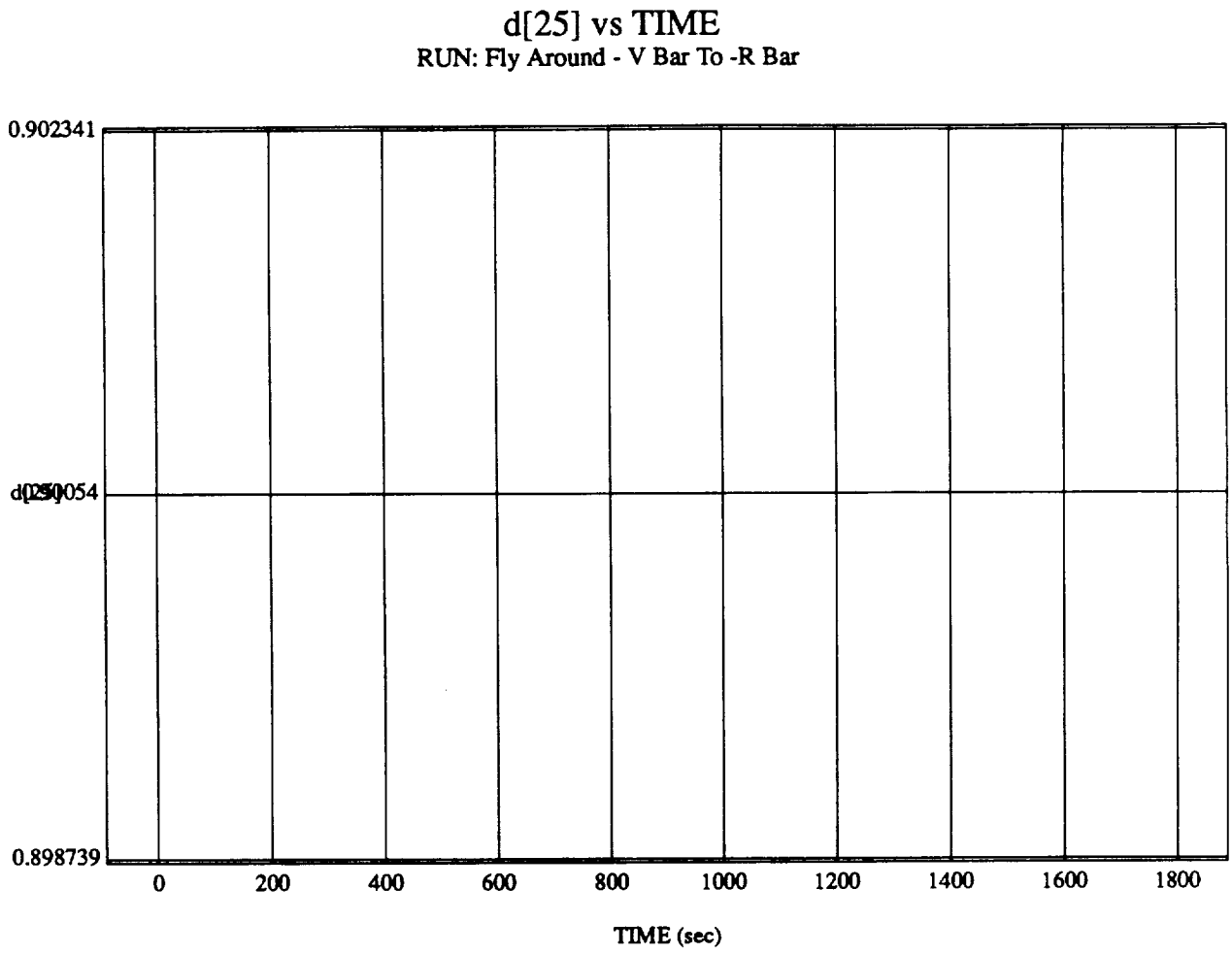
MODULE: ORBITER.lm_elev
DATA SAMPLING FREQUENCY: 0.200 Hz



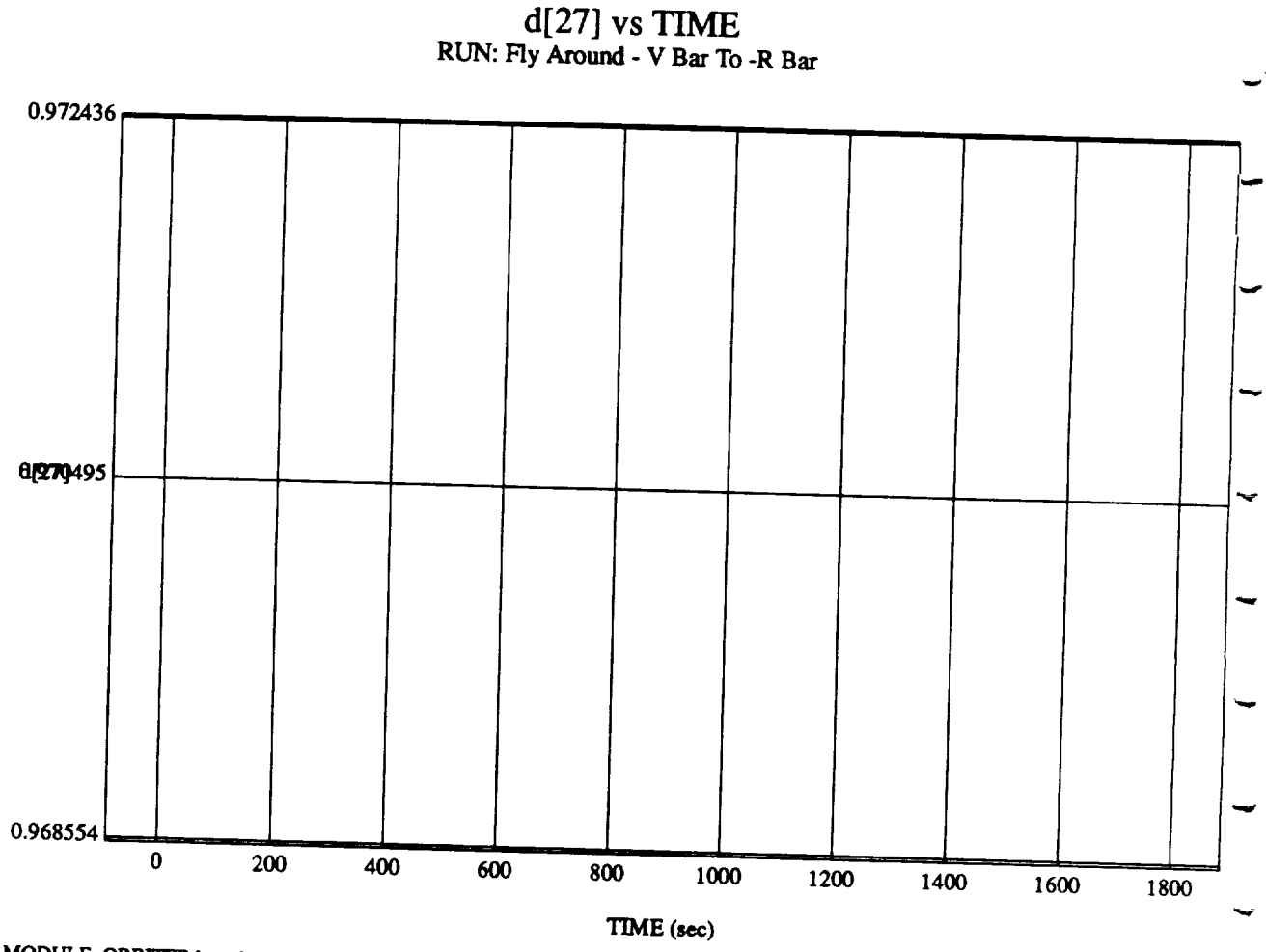
MODULE: ORBITER.lm_elev
DATA SAMPLING FREQUENCY: 0.200 Hz



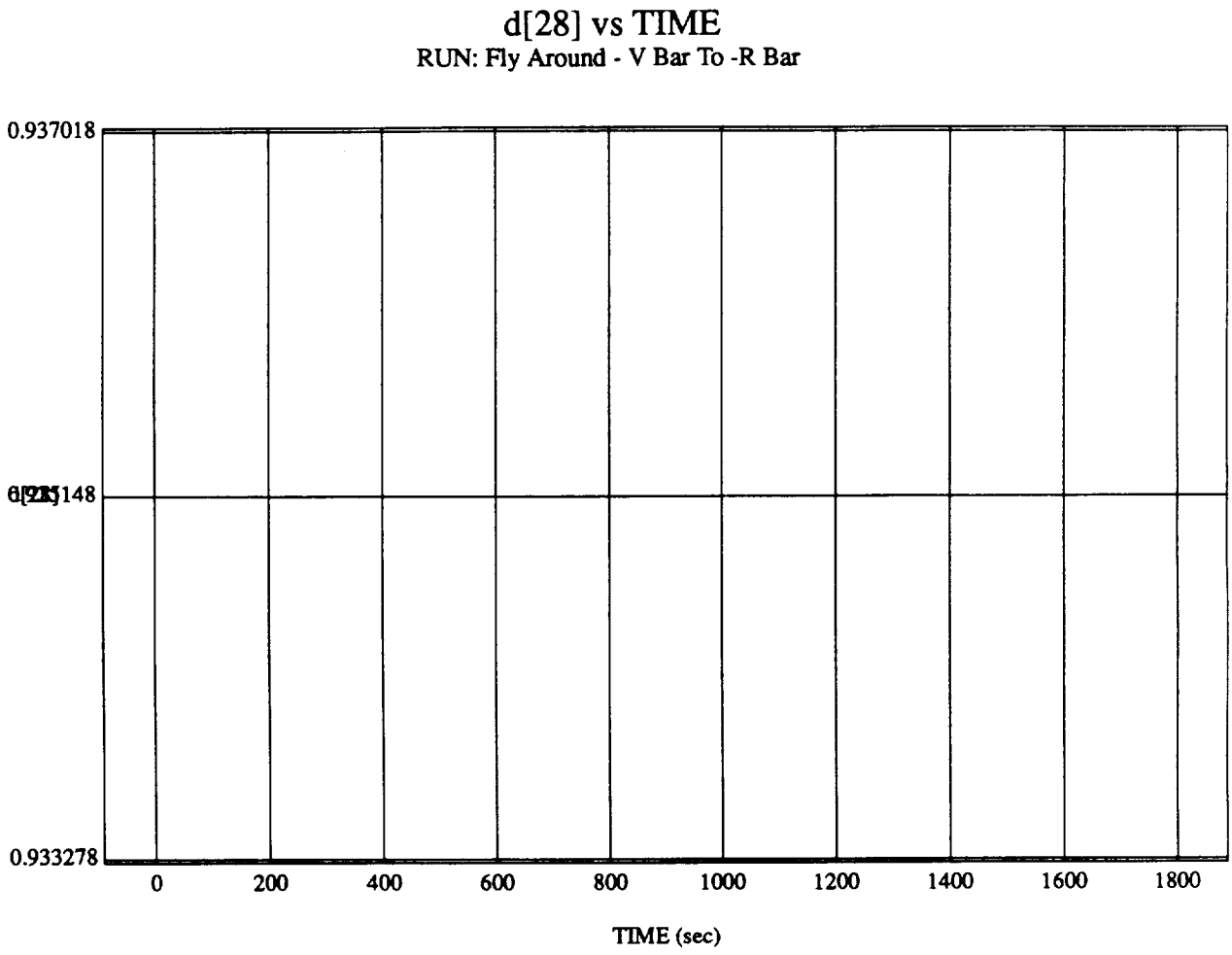
MODULE: ORBITER.lm_elev
DATA SAMPLING FREQUENCY: 0.200 Hz



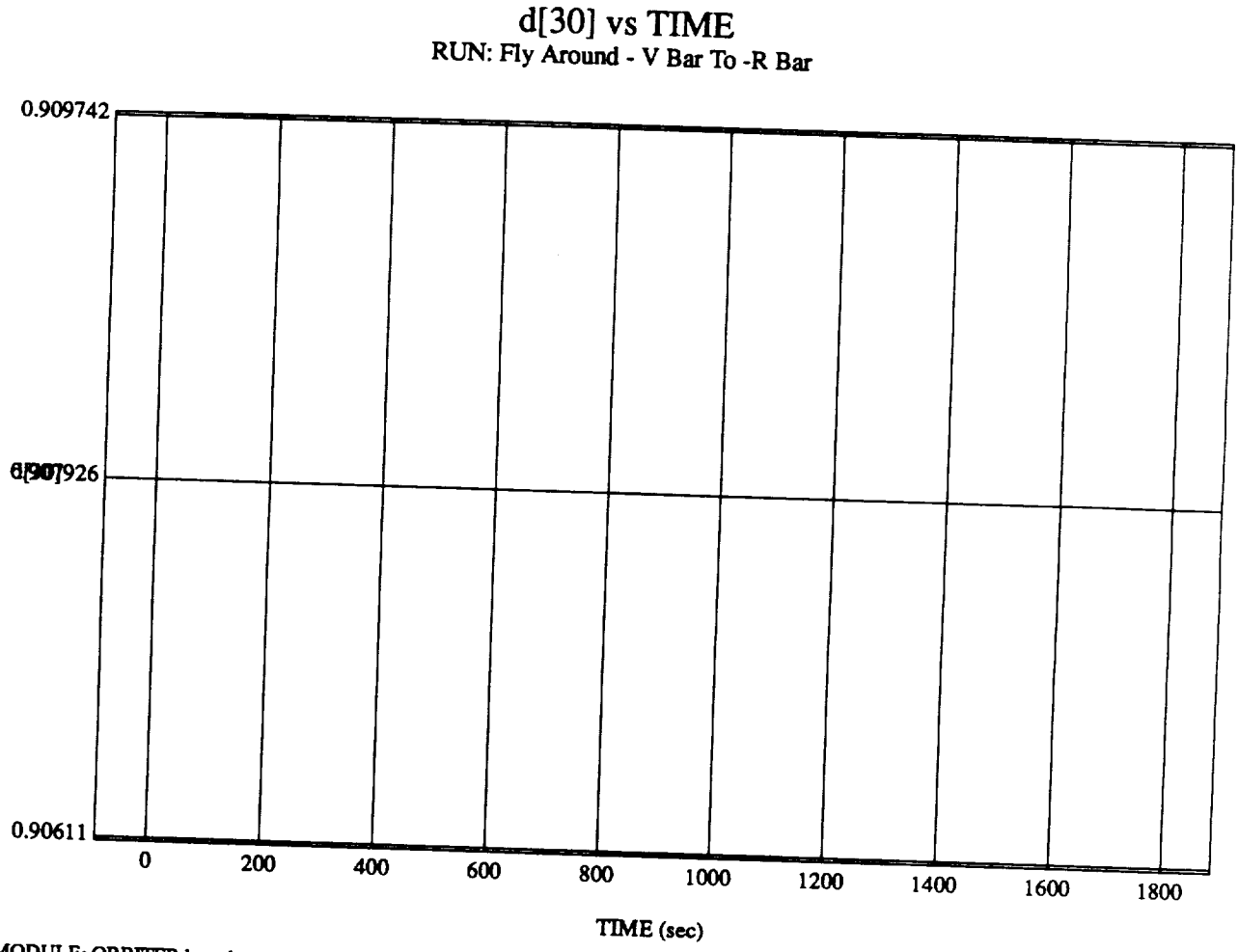
MODULE: ORBITER.lm_elev
DATA SAMPLING FREQUENCY: 0.200 Hz



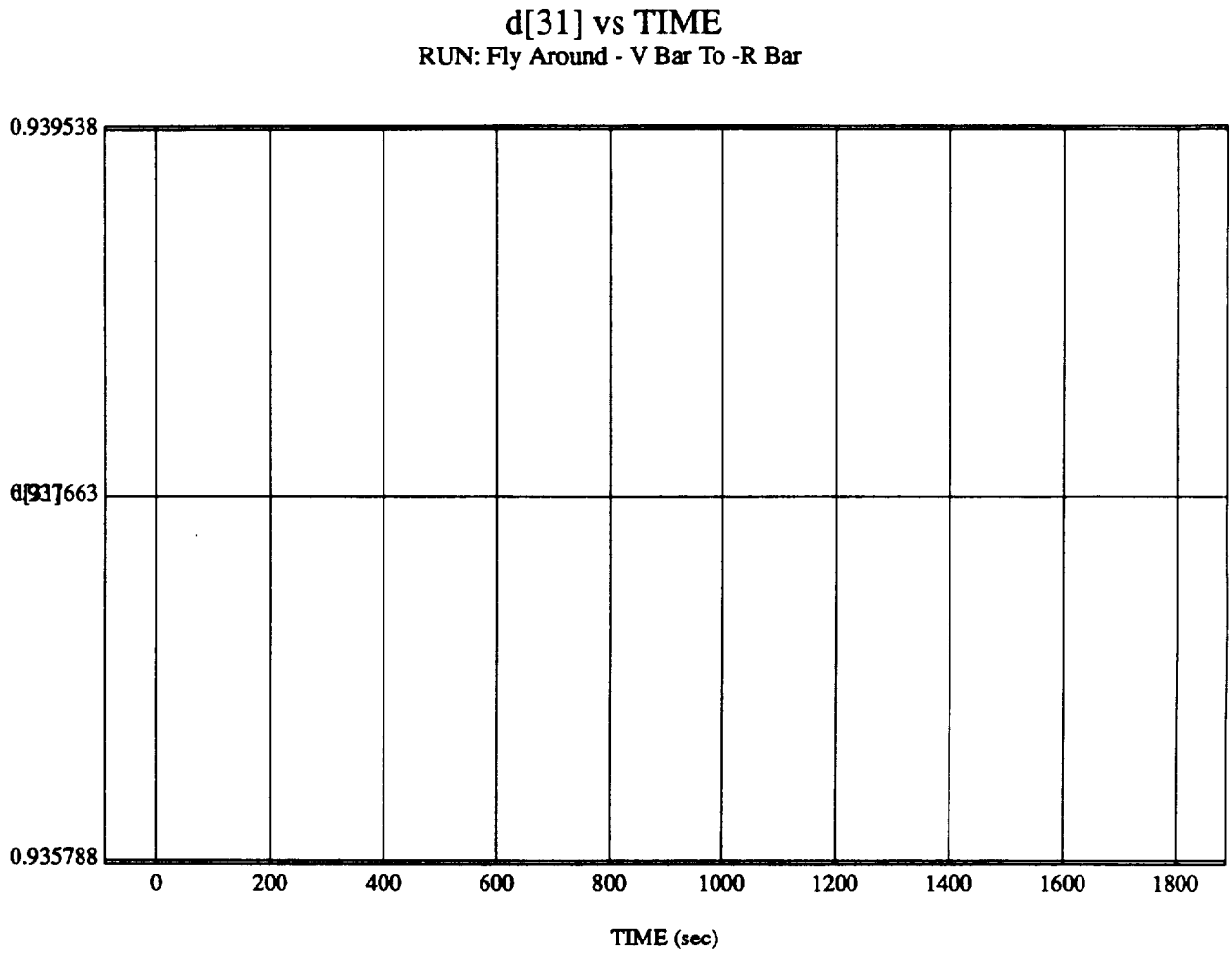
MODULE: ORBITER.lm_elev
DATA SAMPLING FREQUENCY: 0.200 Hz



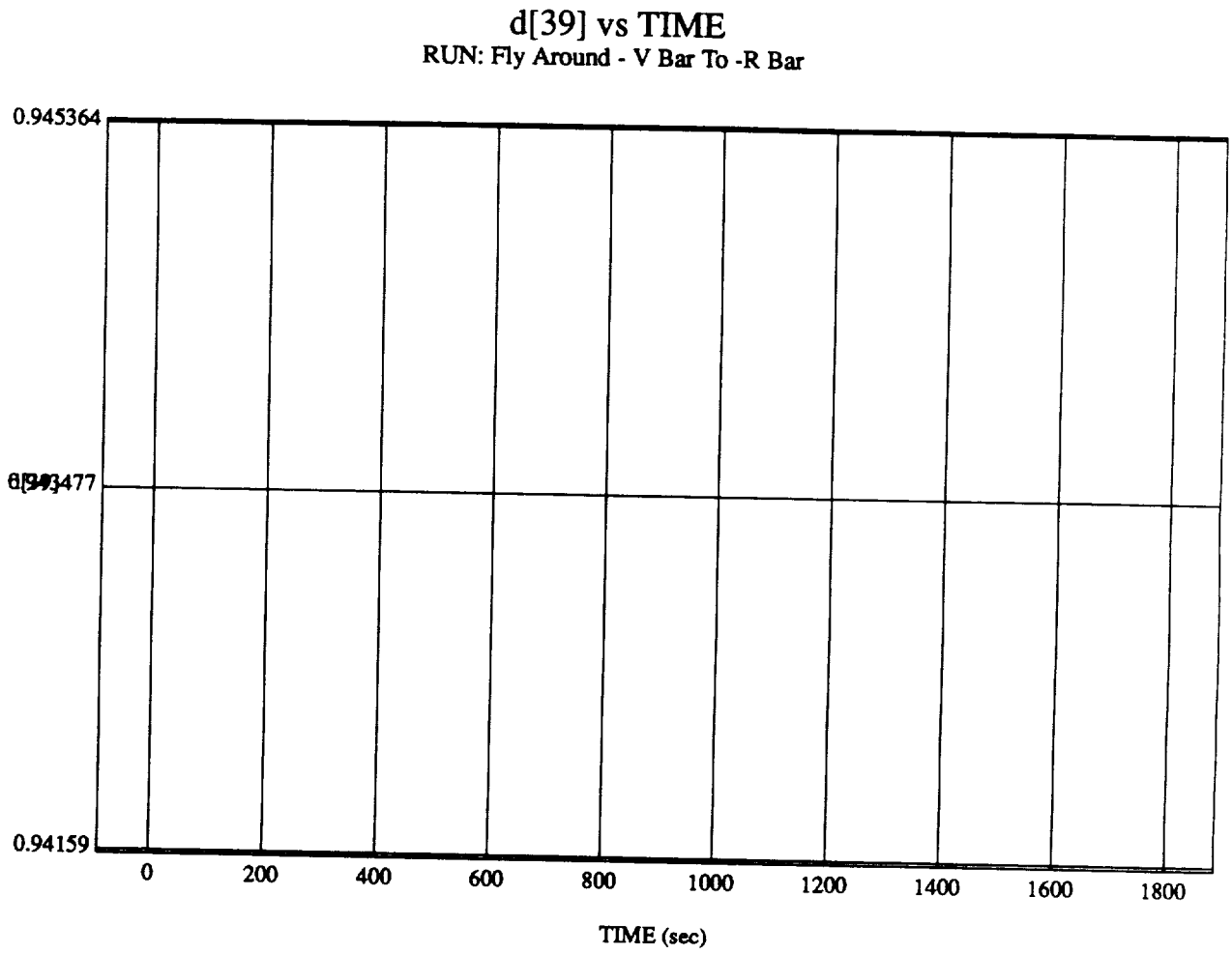
MODULE: ORBITER.lm_elev
DATA SAMPLING FREQUENCY: 0.200 Hz



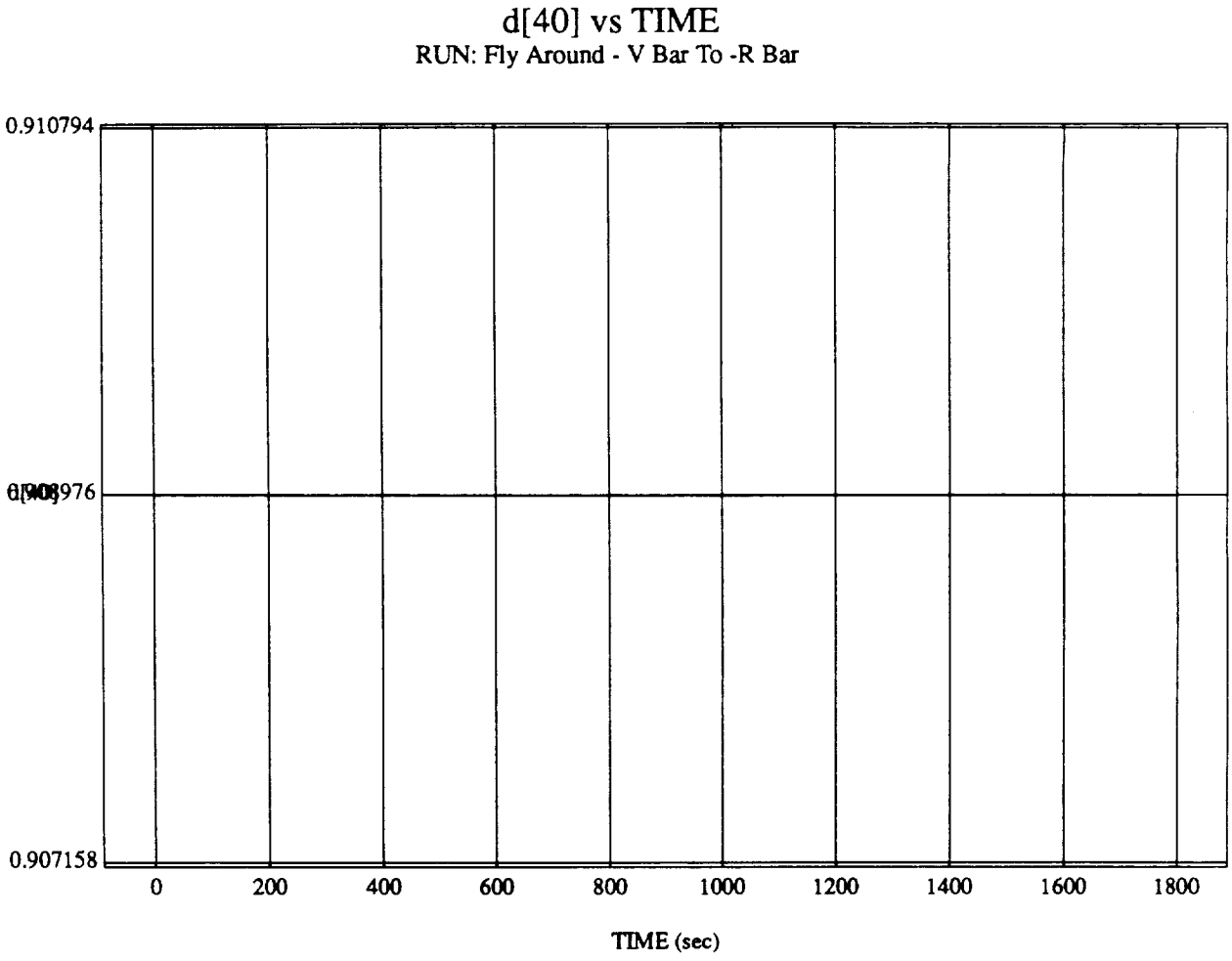
MODULE: ORBITER.lm_elev
DATA SAMPLING FREQUENCY: 0.200 Hz



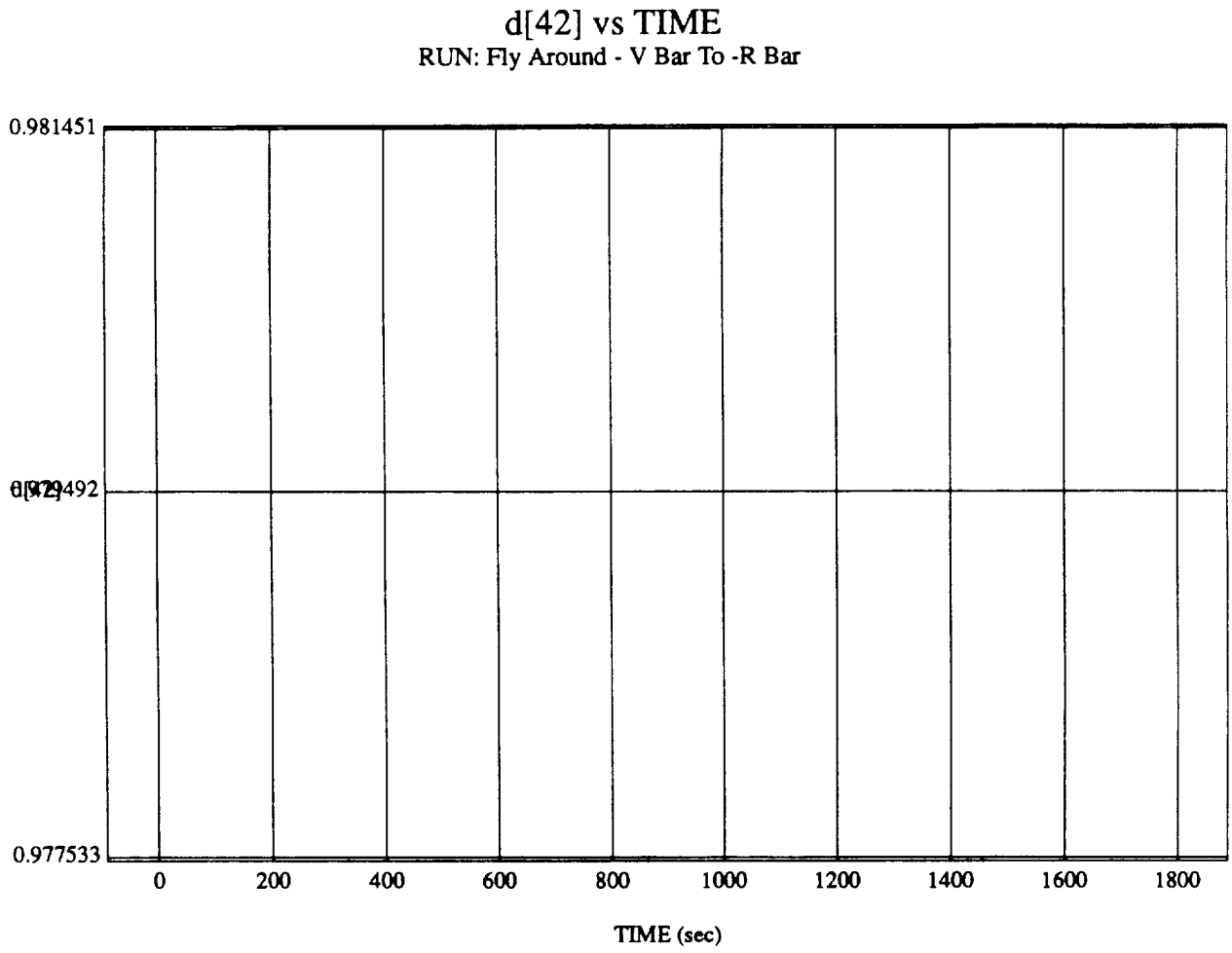
MODULE: ORBITER.lm_elev
DATA SAMPLING FREQUENCY: 0.200 Hz



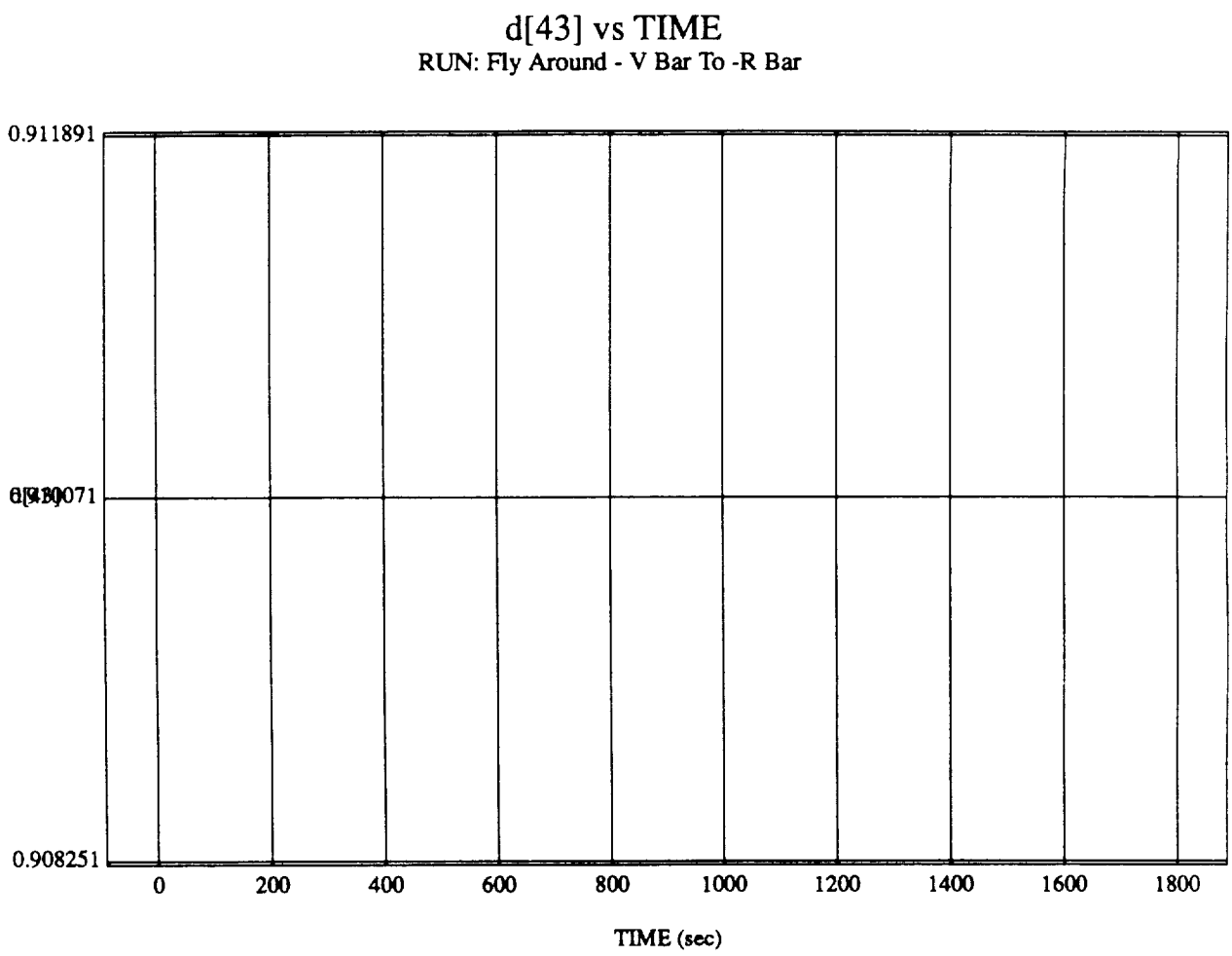
MODULE: ORBITER.lm_elev
DATA SAMPLING FREQUENCY: 0.200 Hz



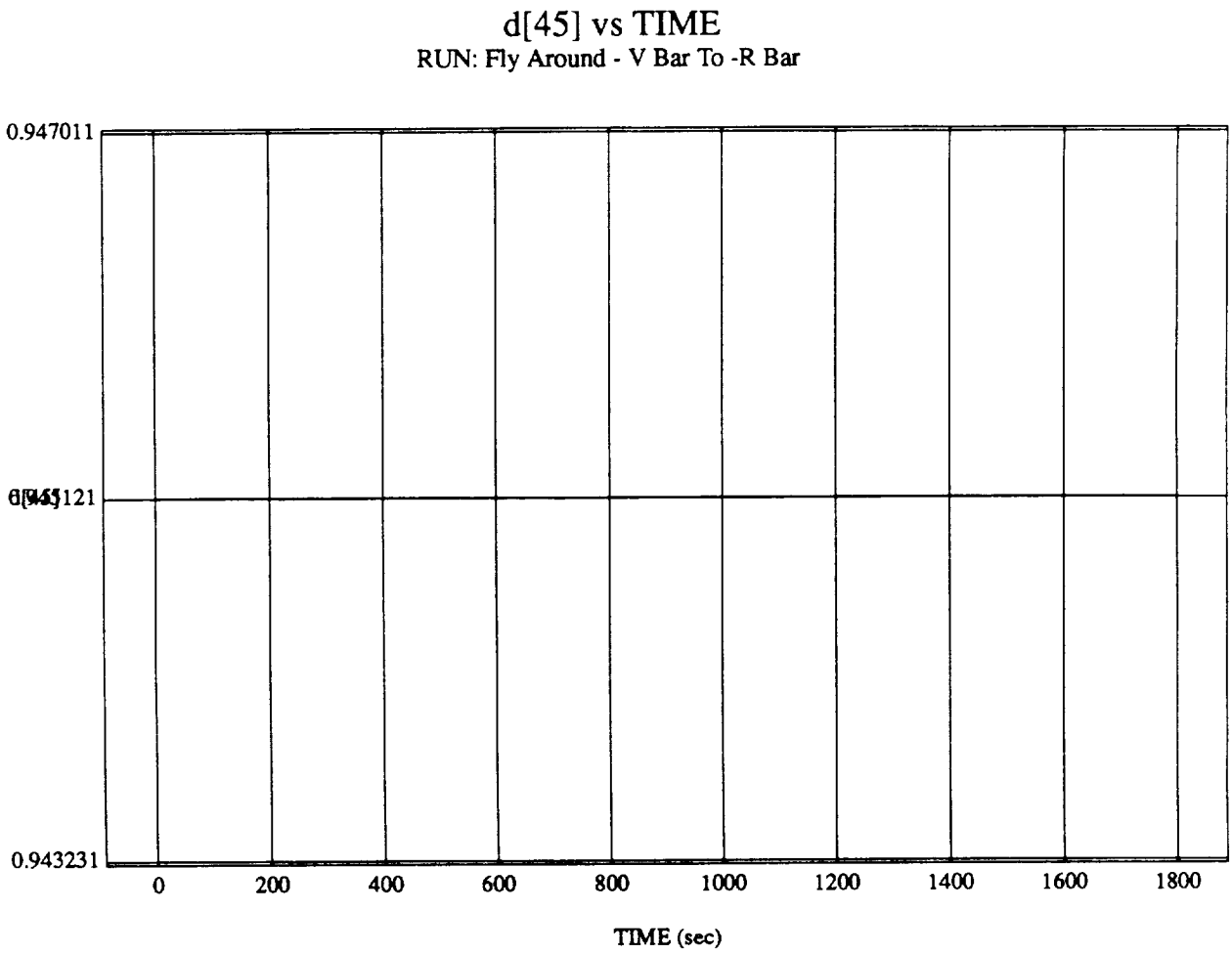
MODULE: ORBITER.lm_elev
DATA SAMPLING FREQUENCY: 0.200 Hz



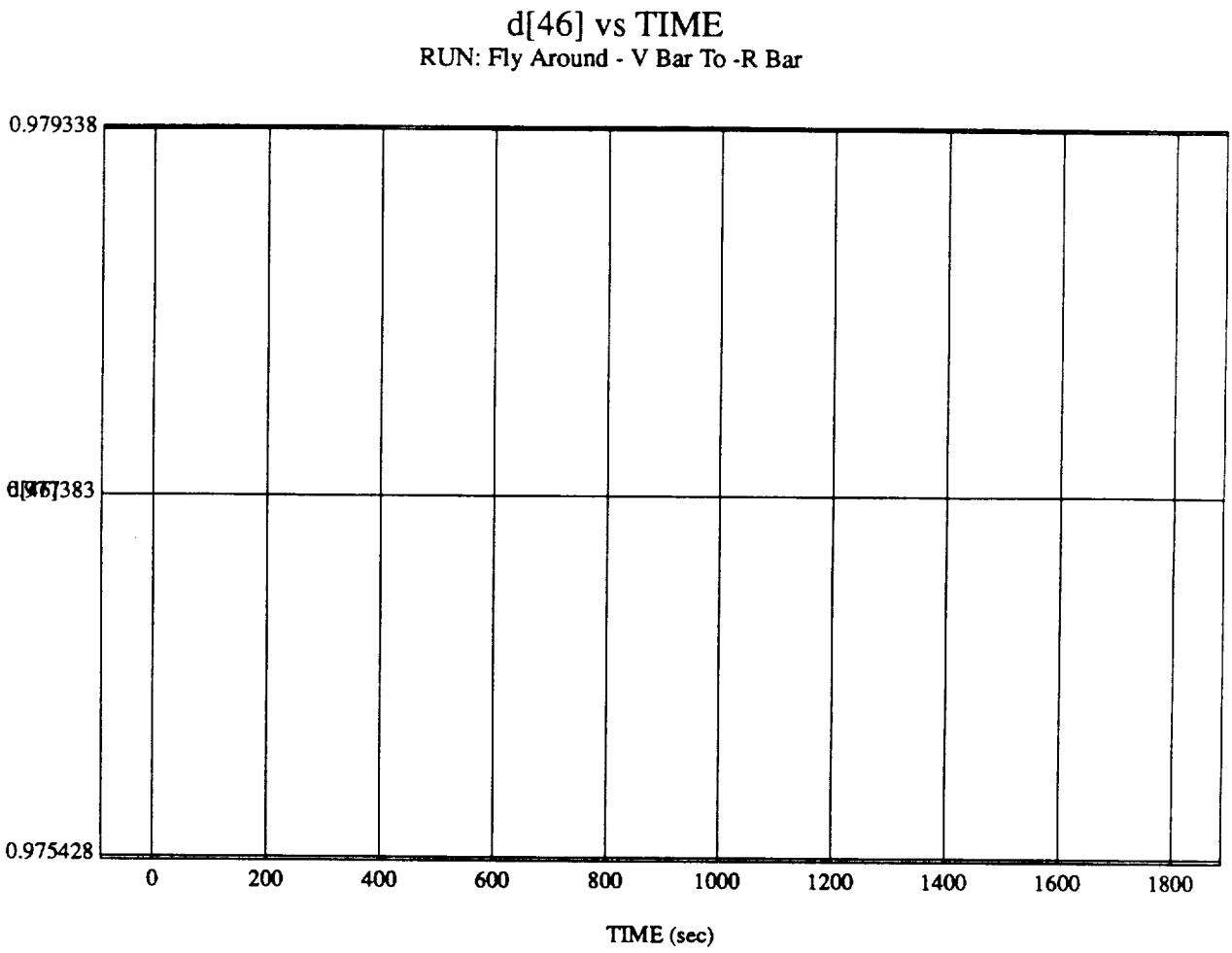
MODULE: ORBITER.lm_elev
DATA SAMPLING FREQUENCY: 0.200 Hz



MODULE: ORBITER.lm_elev
DATA SAMPLING FREQUENCY: 0.200 Hz



MODULE: ORBITER.lm_elev
DATA SAMPLING FREQUENCY: 0.200 Hz

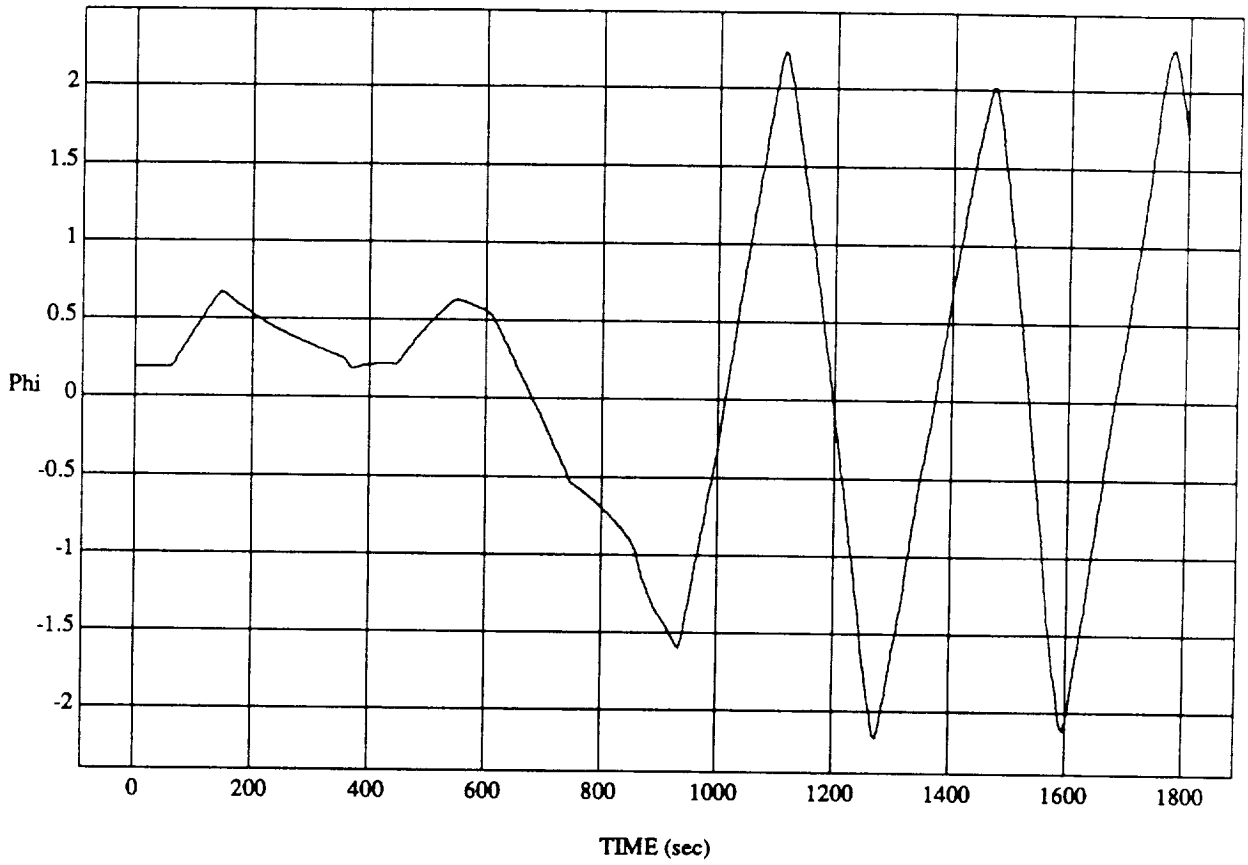


MODULE: ORBITER.lm_elev
DATA SAMPLING FREQUENCY: 0.200 Hz

SIMULATION APPLICATION: ARIC Translational Controller Simulation

Phi vs TIME

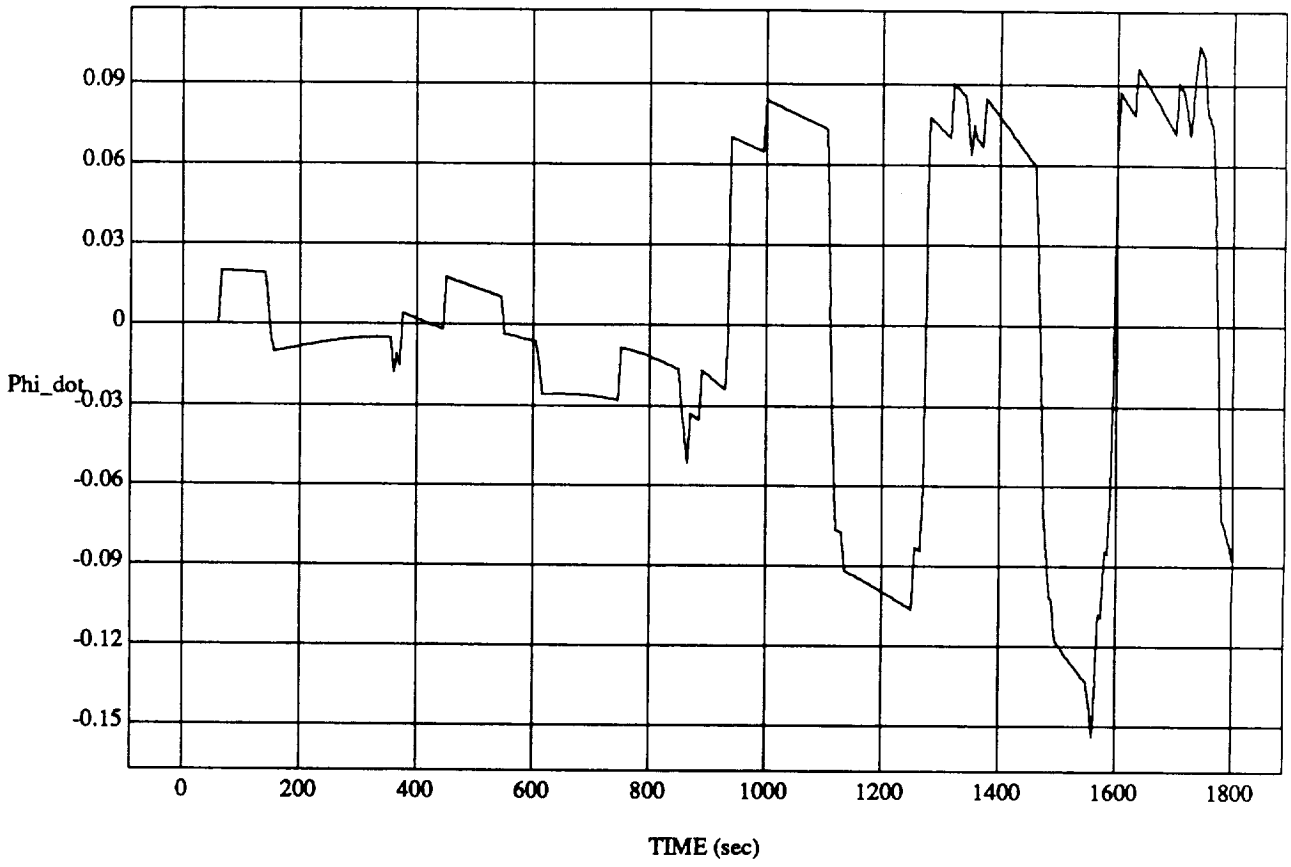
RUN: Fly Around - V Bar To -R Bar



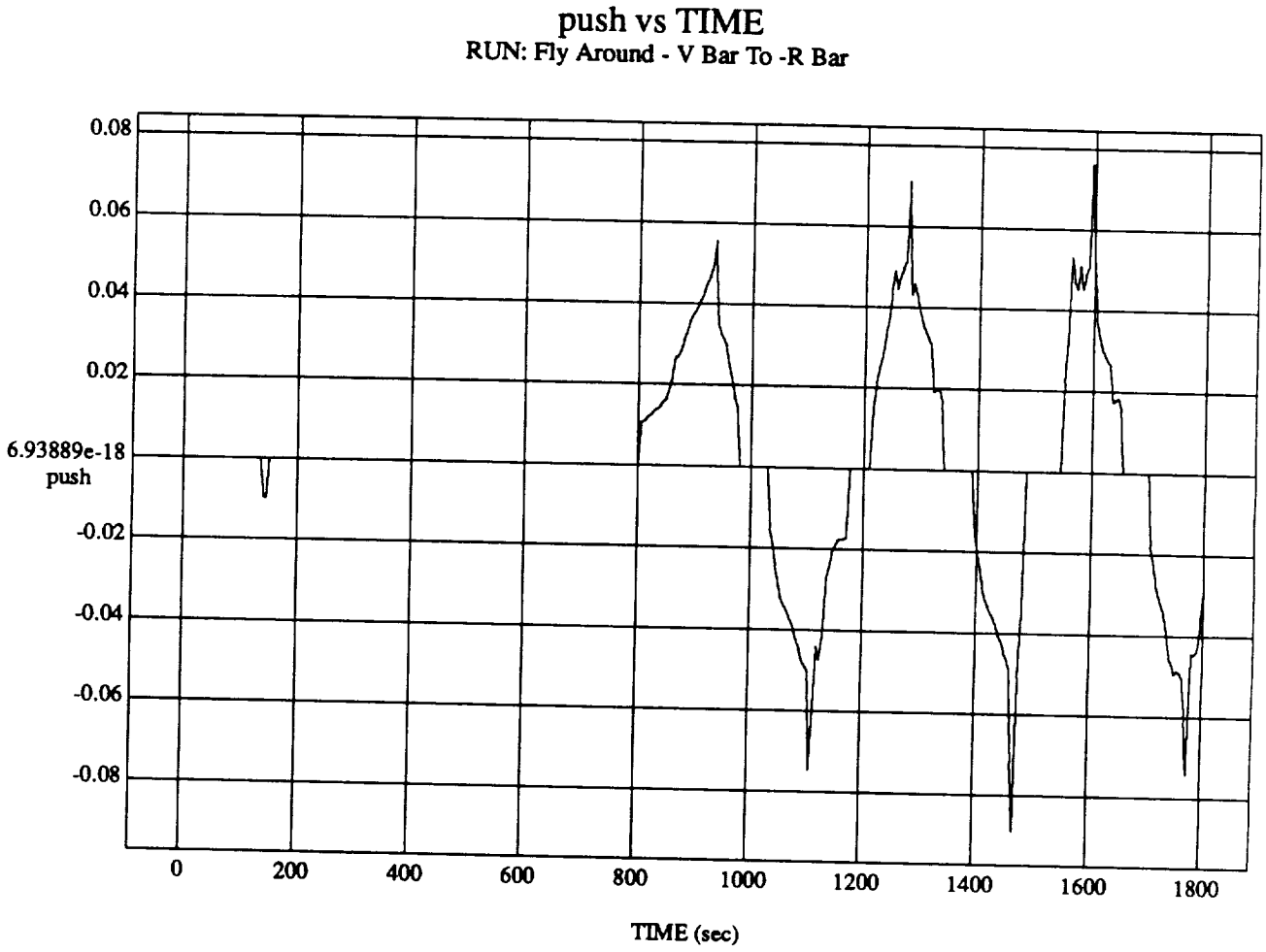
MODULE: ORBITER.lm_range
DATA SAMPLING FREQUENCY: 0.200 Hz

Phi_dot vs TIME

RUN: Fly Around - V Bar To -R Bar



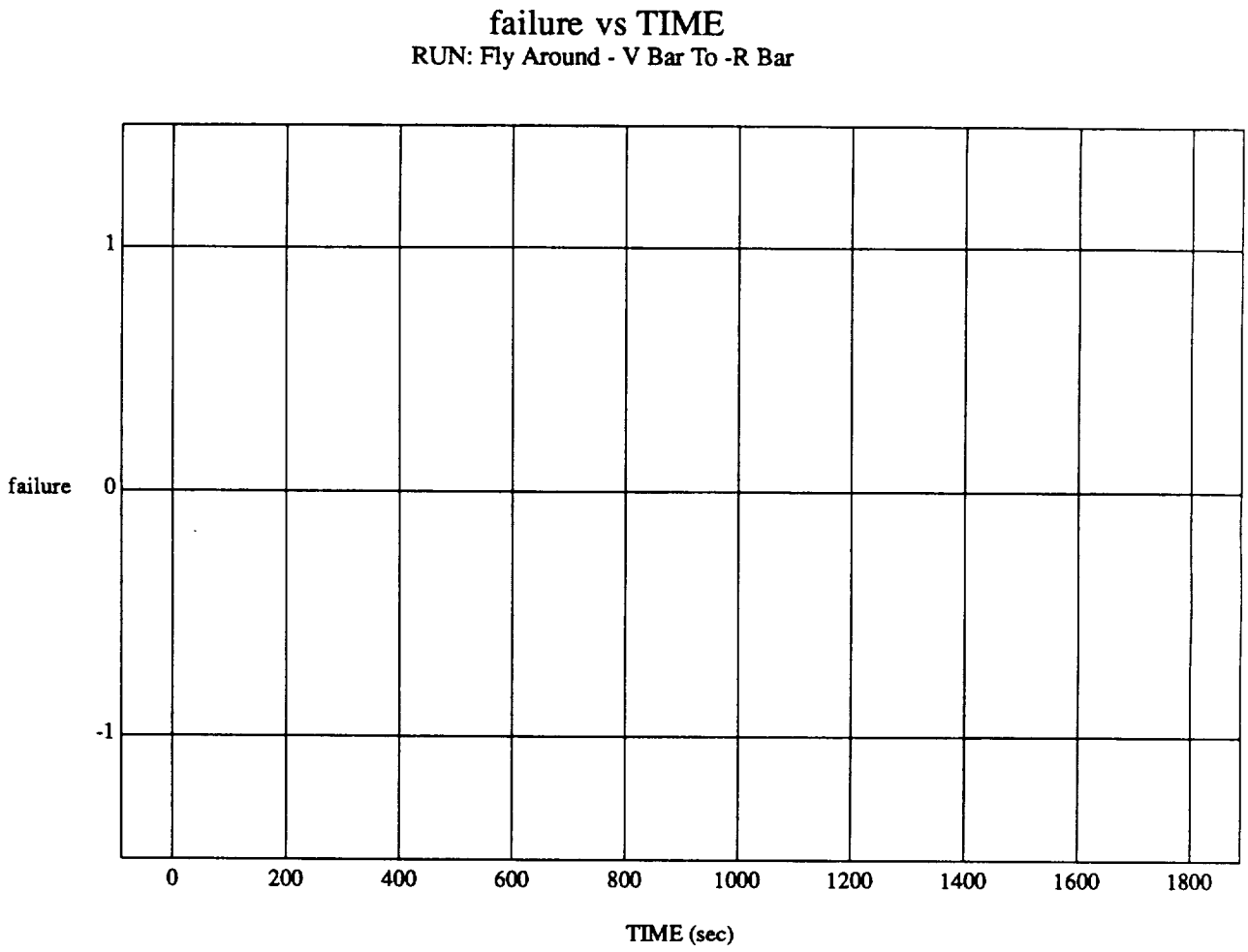
MODULE: ORBITER.lm_range
DATA SAMPLING FREQUENCY: 0.200 Hz



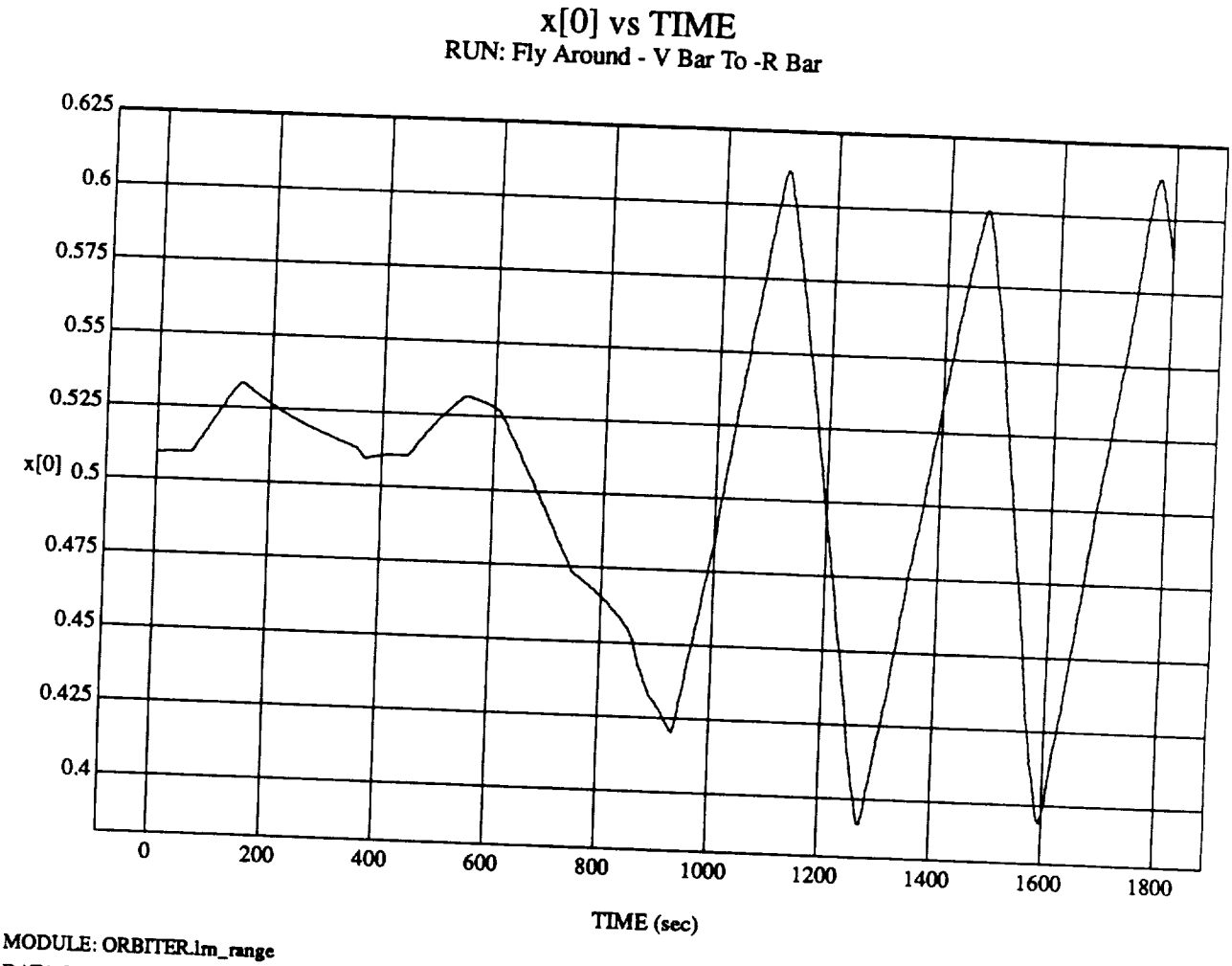
MODULE: ORBITER.lm_range
DATA SAMPLING FREQUENCY: 0.200 Hz

273

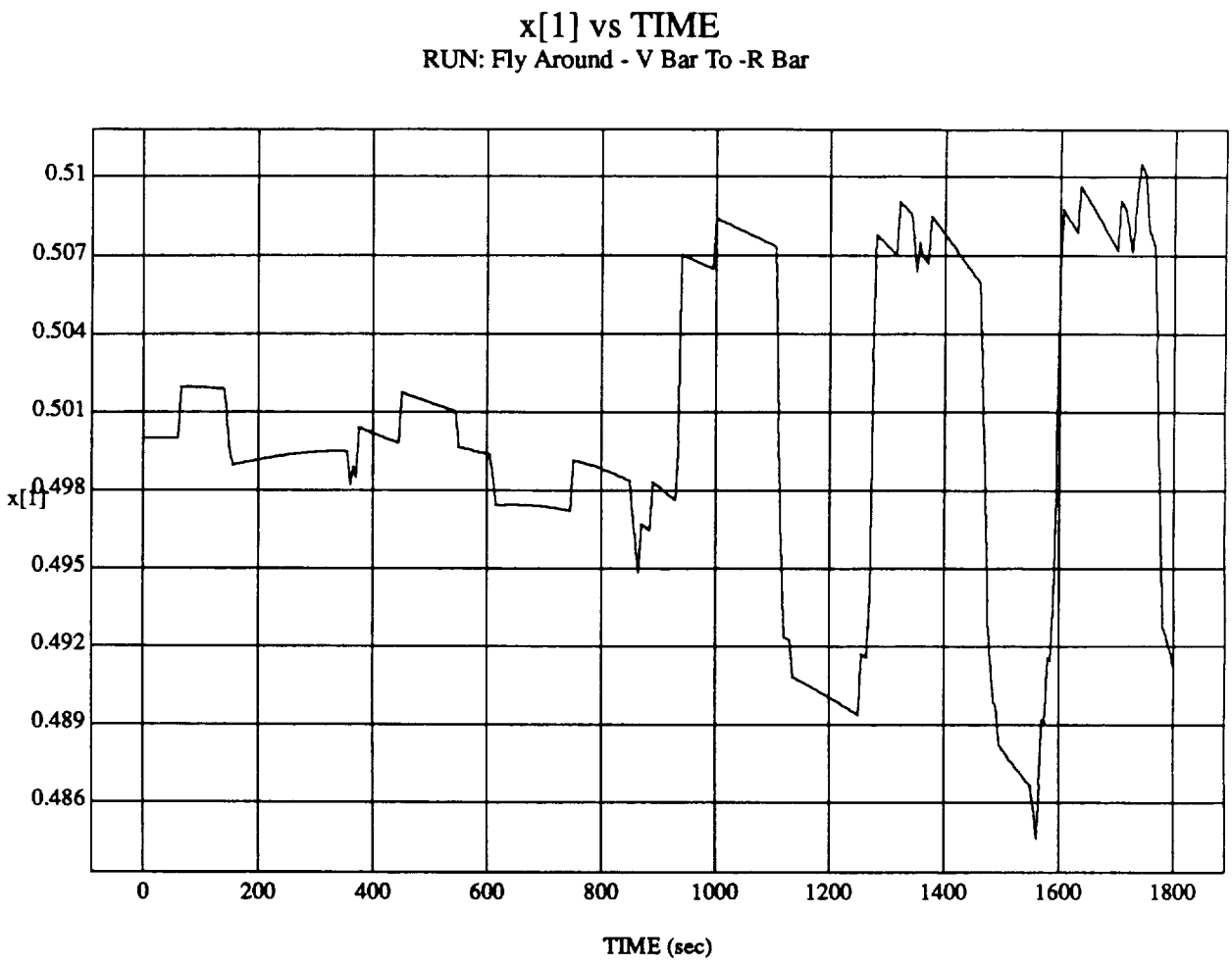
SIMULATION APPLICATION: ARIC Translational Controller Simulation



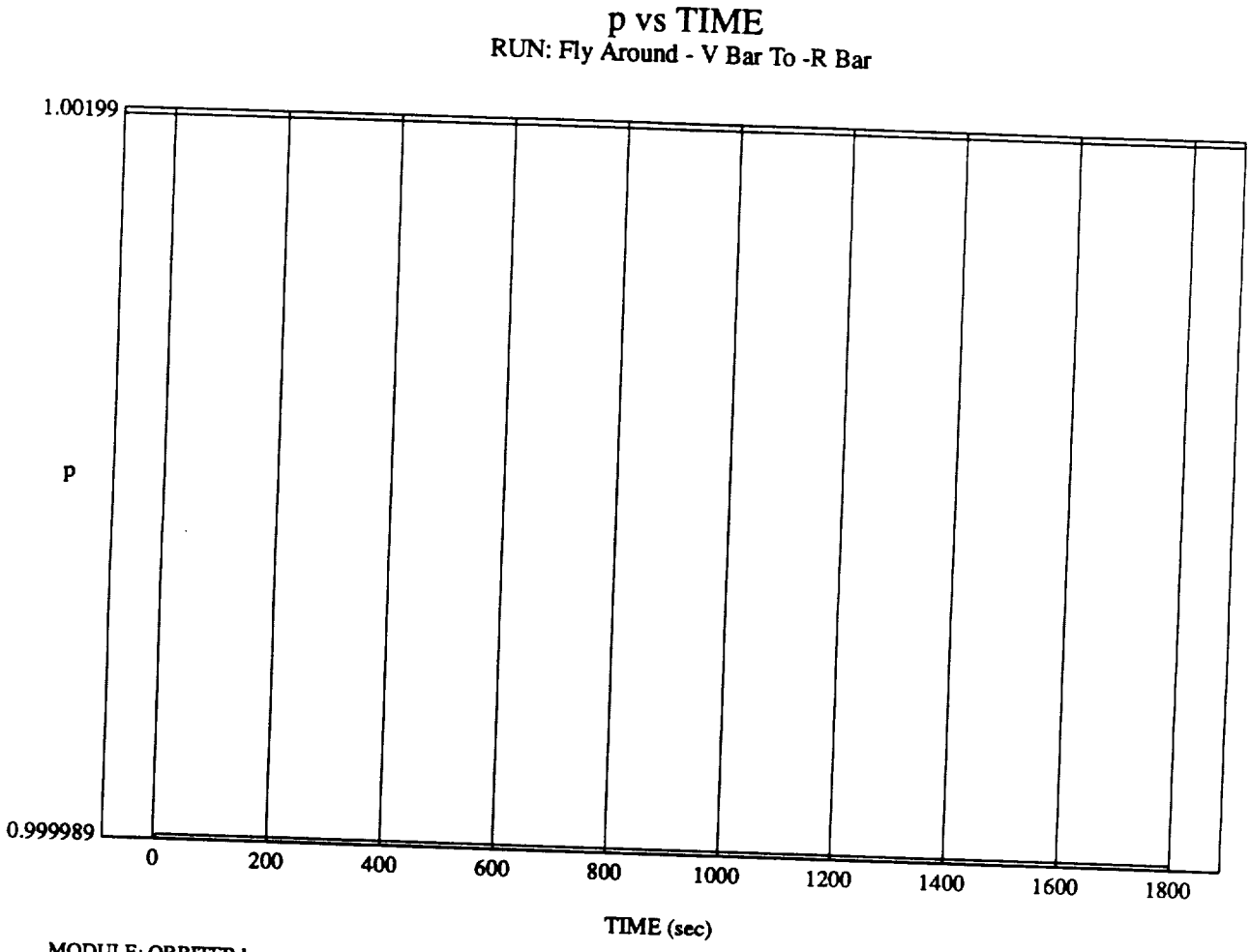
MODULE: ORBITER.lm_range
DATA SAMPLING FREQUENCY: 0.200 Hz



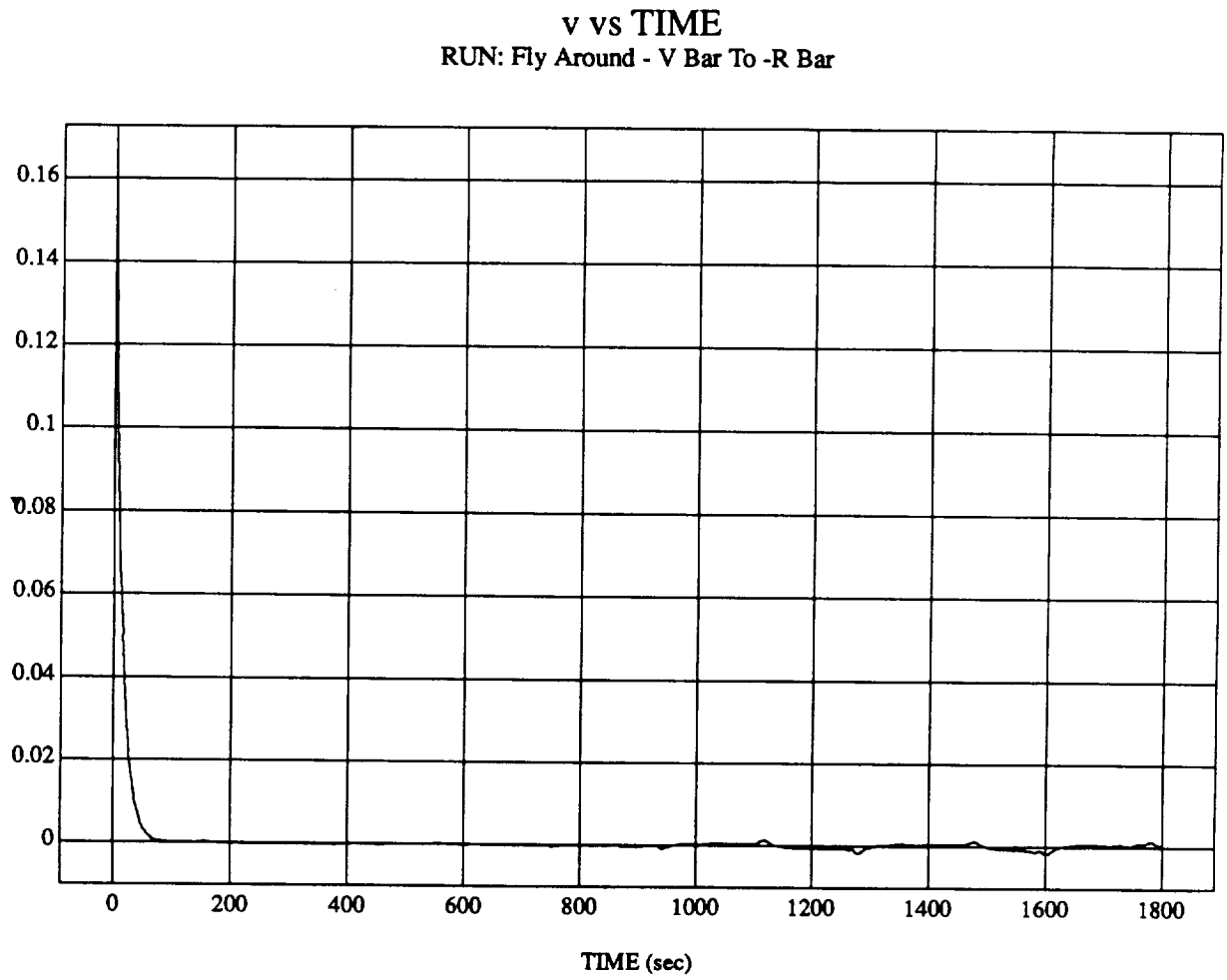
MODULE: ORBITER.lm_range
DATA SAMPLING FREQUENCY: 0.200 Hz



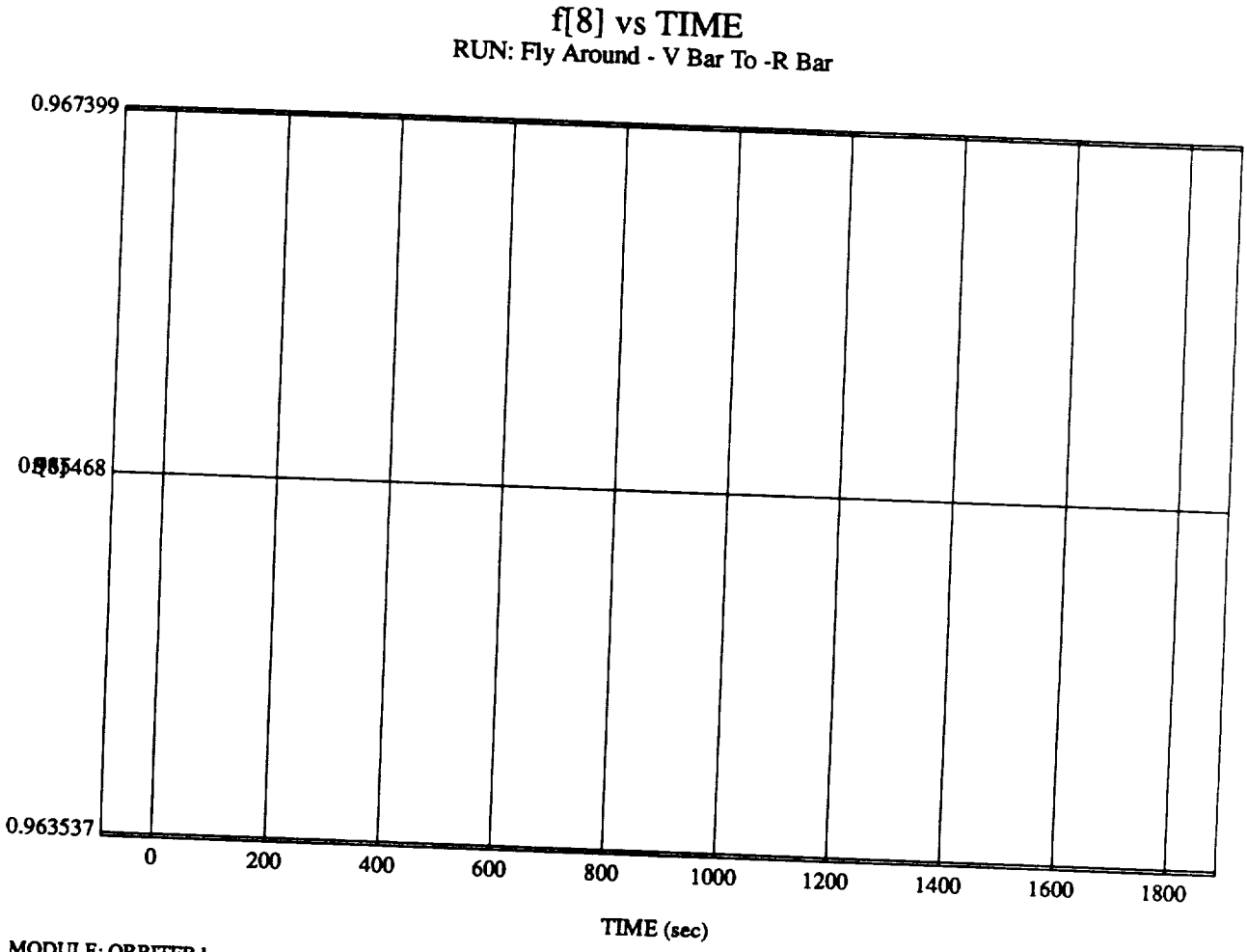
MODULE: ORBITER.lm_range
DATA SAMPLING FREQUENCY: 0.200 Hz



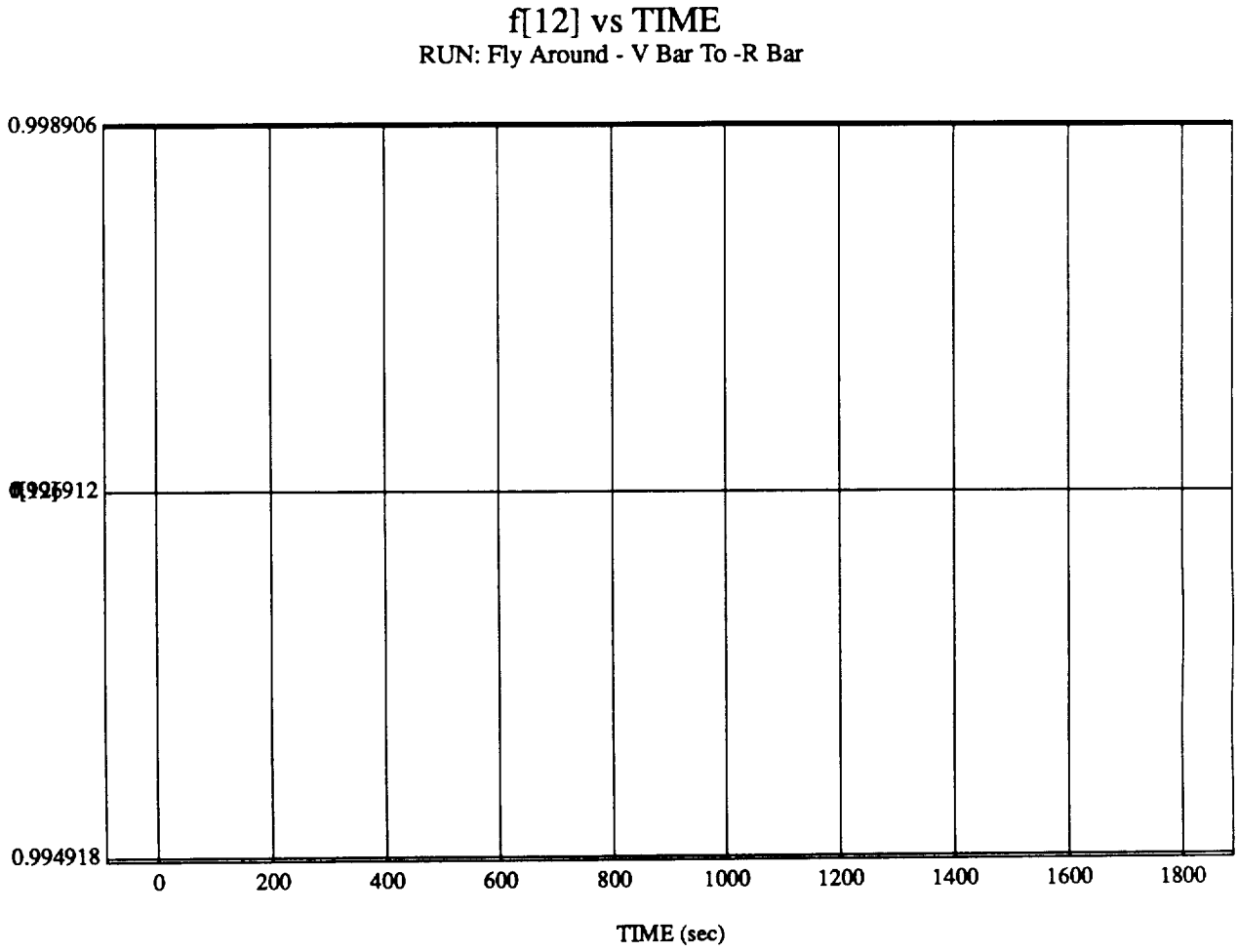
MODULE: ORBITER.lm_range
DATA SAMPLING FREQUENCY: 0.200 Hz



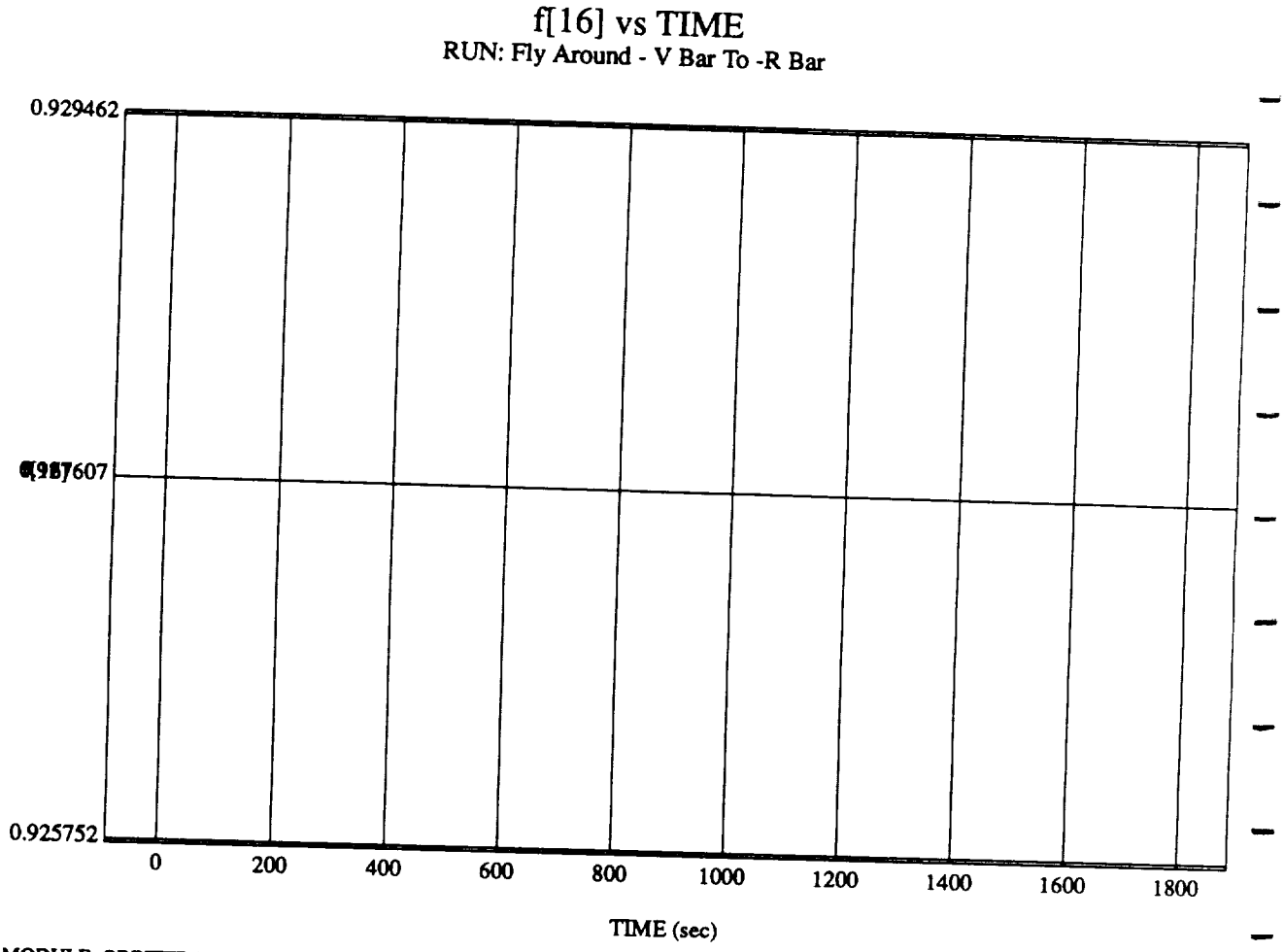
MODULE: ORBITER.lm_range
DATA SAMPLING FREQUENCY: 0.200 Hz



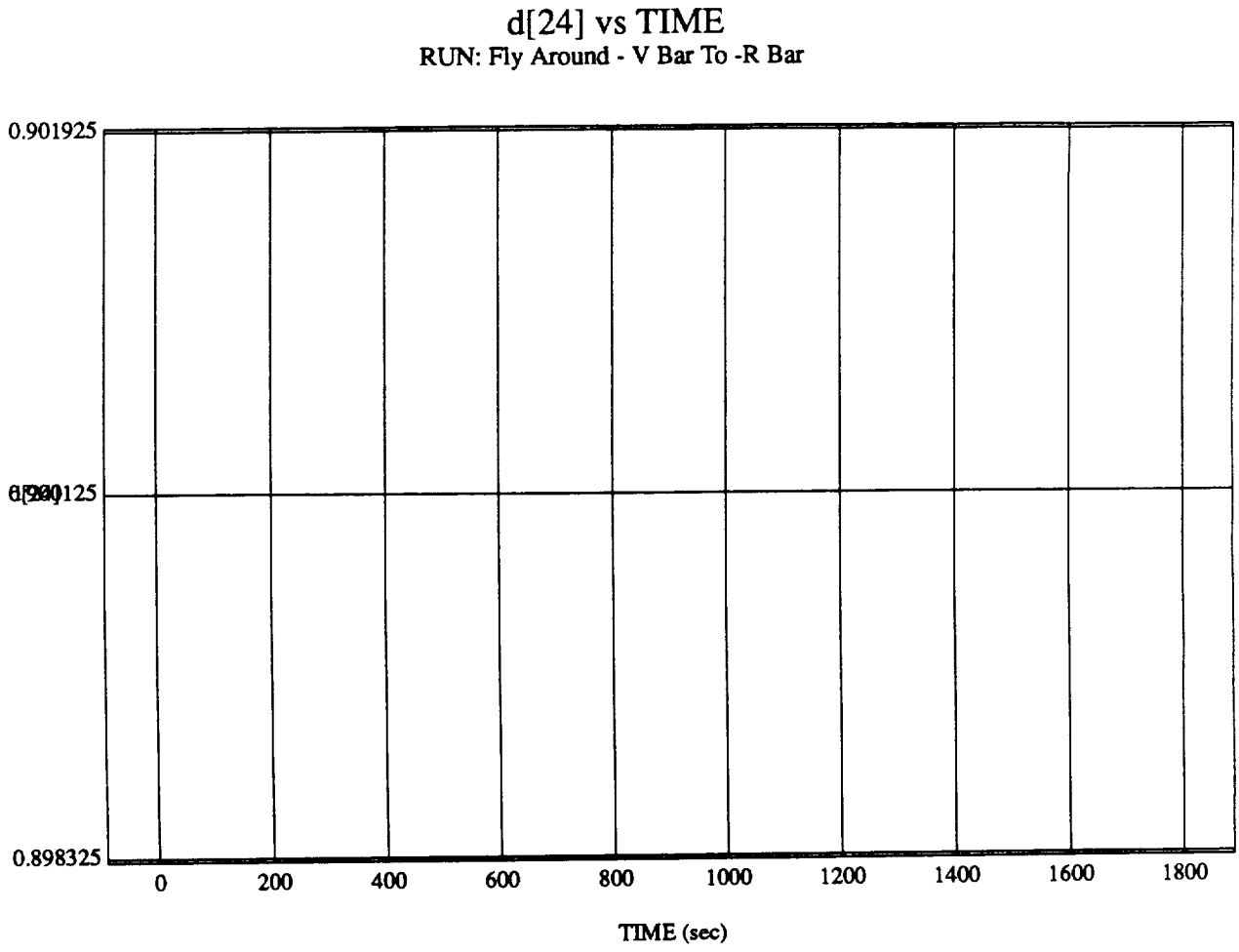
MODULE: ORBITER.lm_range
DATA SAMPLING FREQUENCY: 0.200 Hz



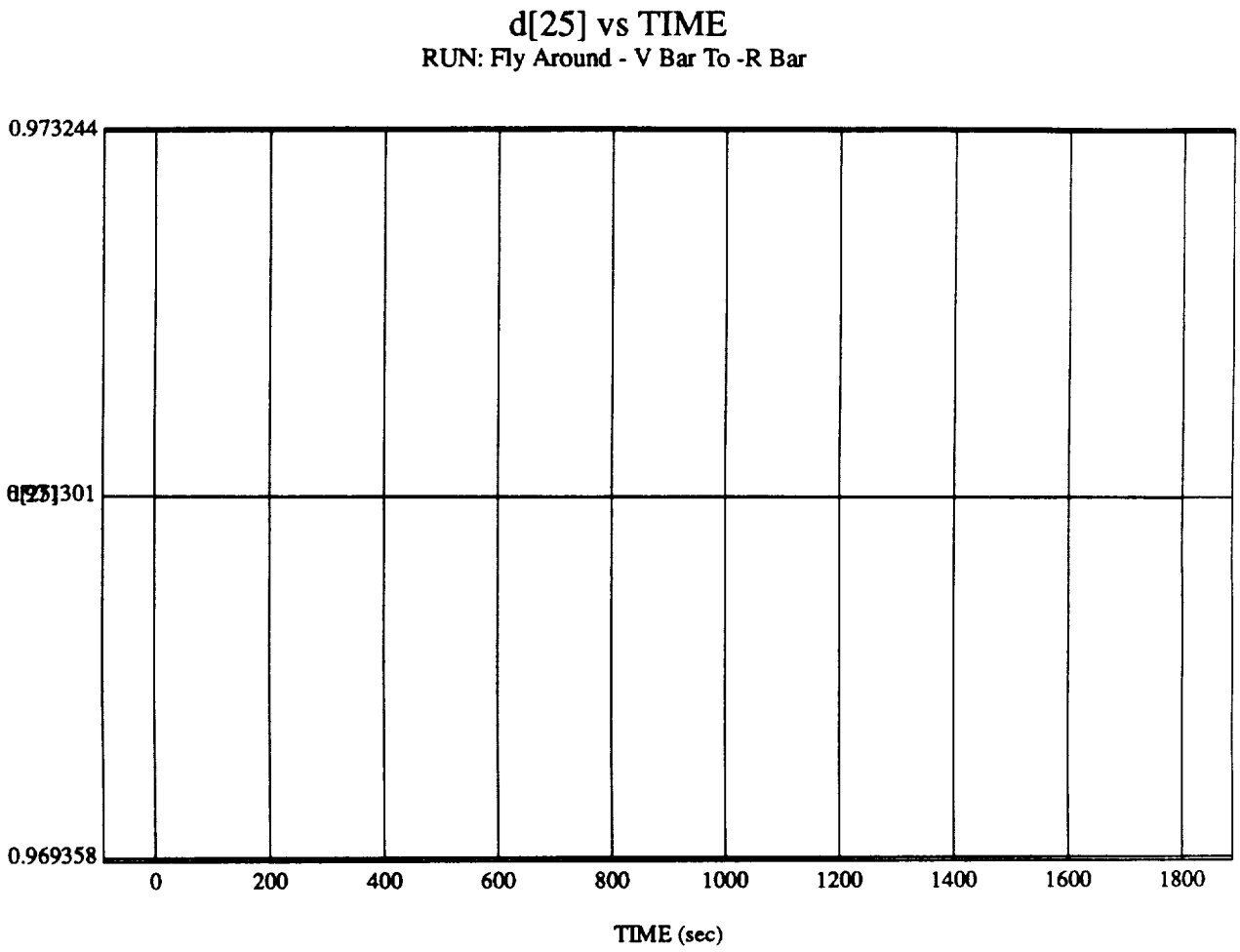
MODULE: ORBITER.lm_range
DATA SAMPLING FREQUENCY: 0.200 Hz



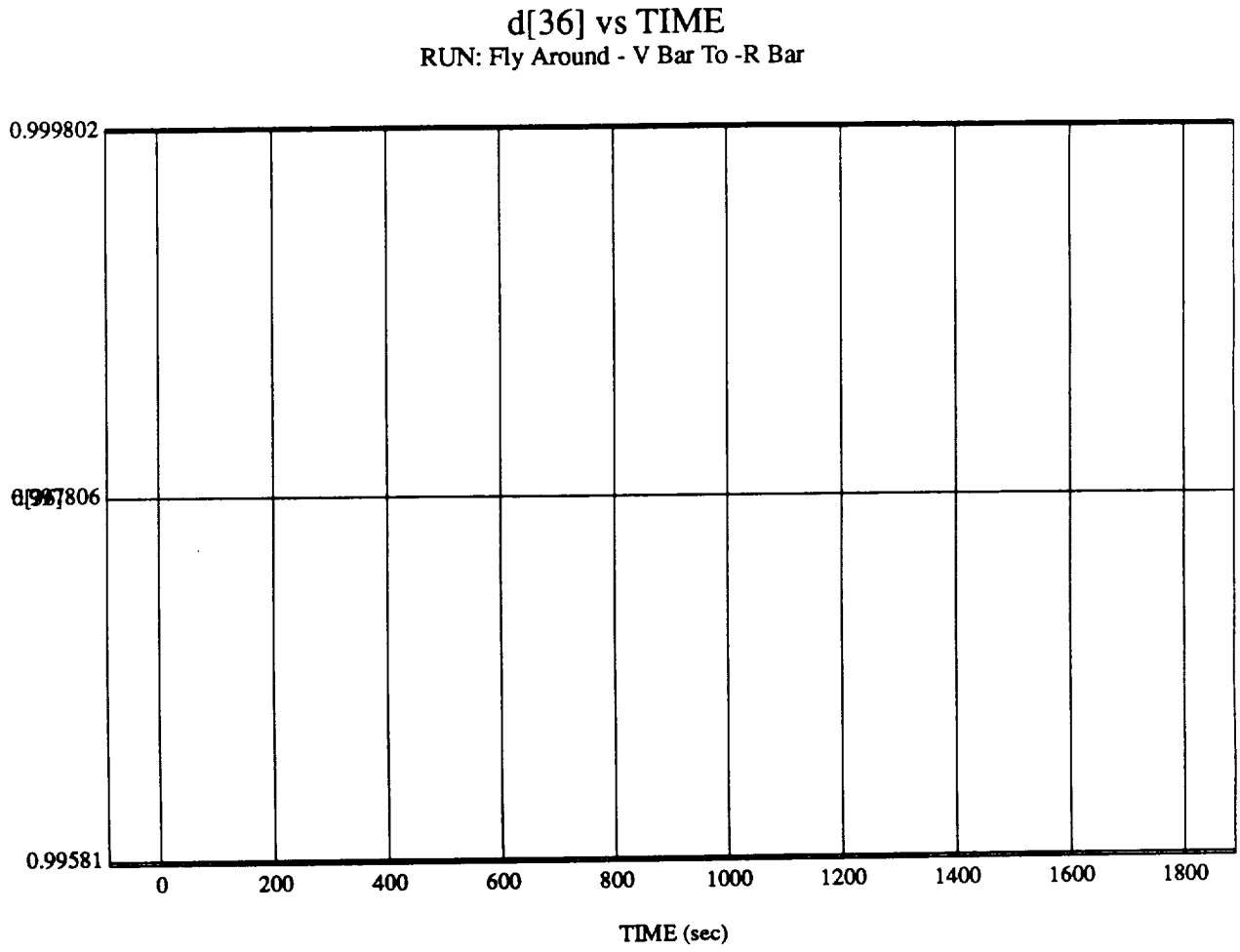
MODULE: ORBITER.lm_range
DATA SAMPLING FREQUENCY: 0.200 Hz



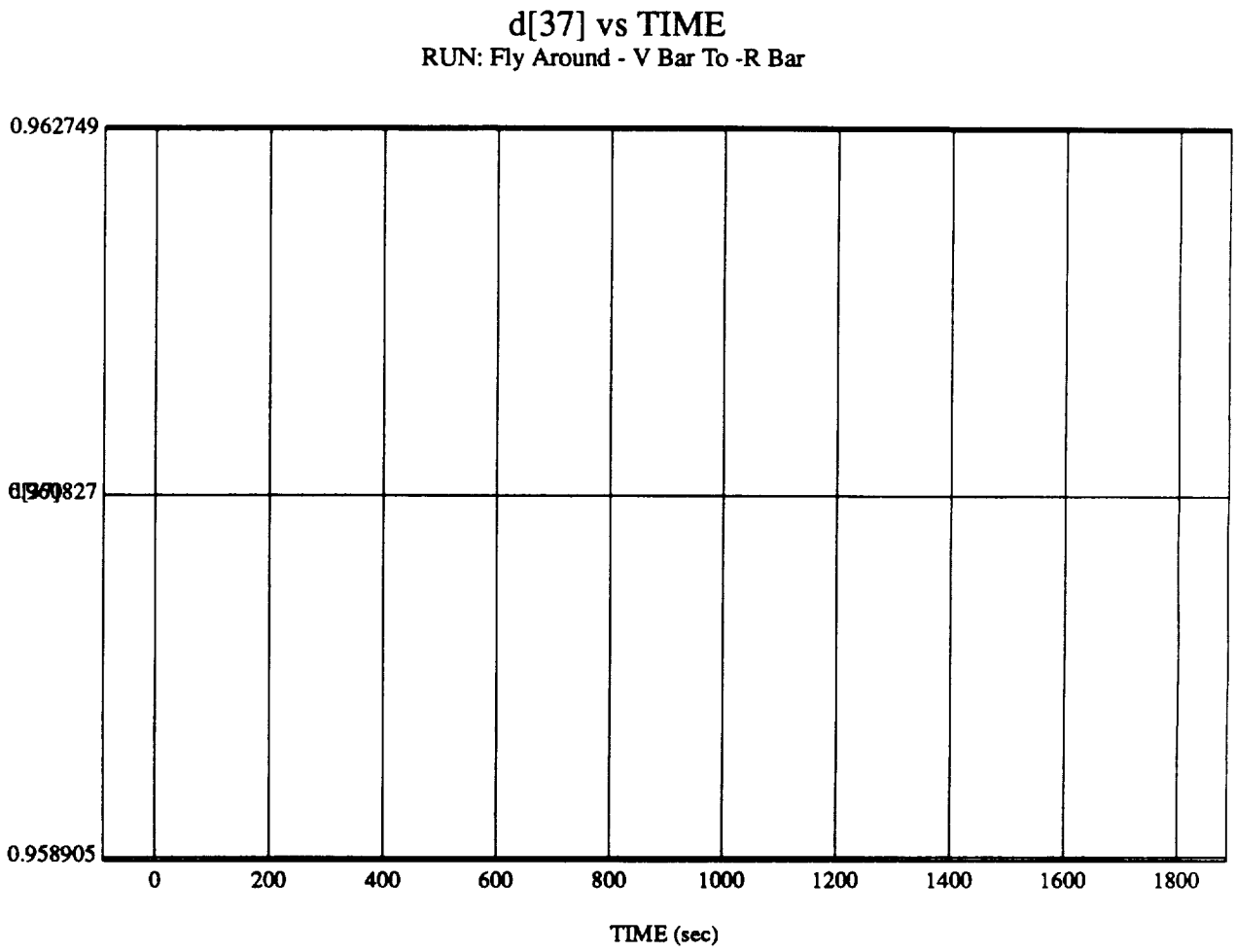
MODULE: ORBITER.lm_range
DATA SAMPLING FREQUENCY: 0.200 Hz



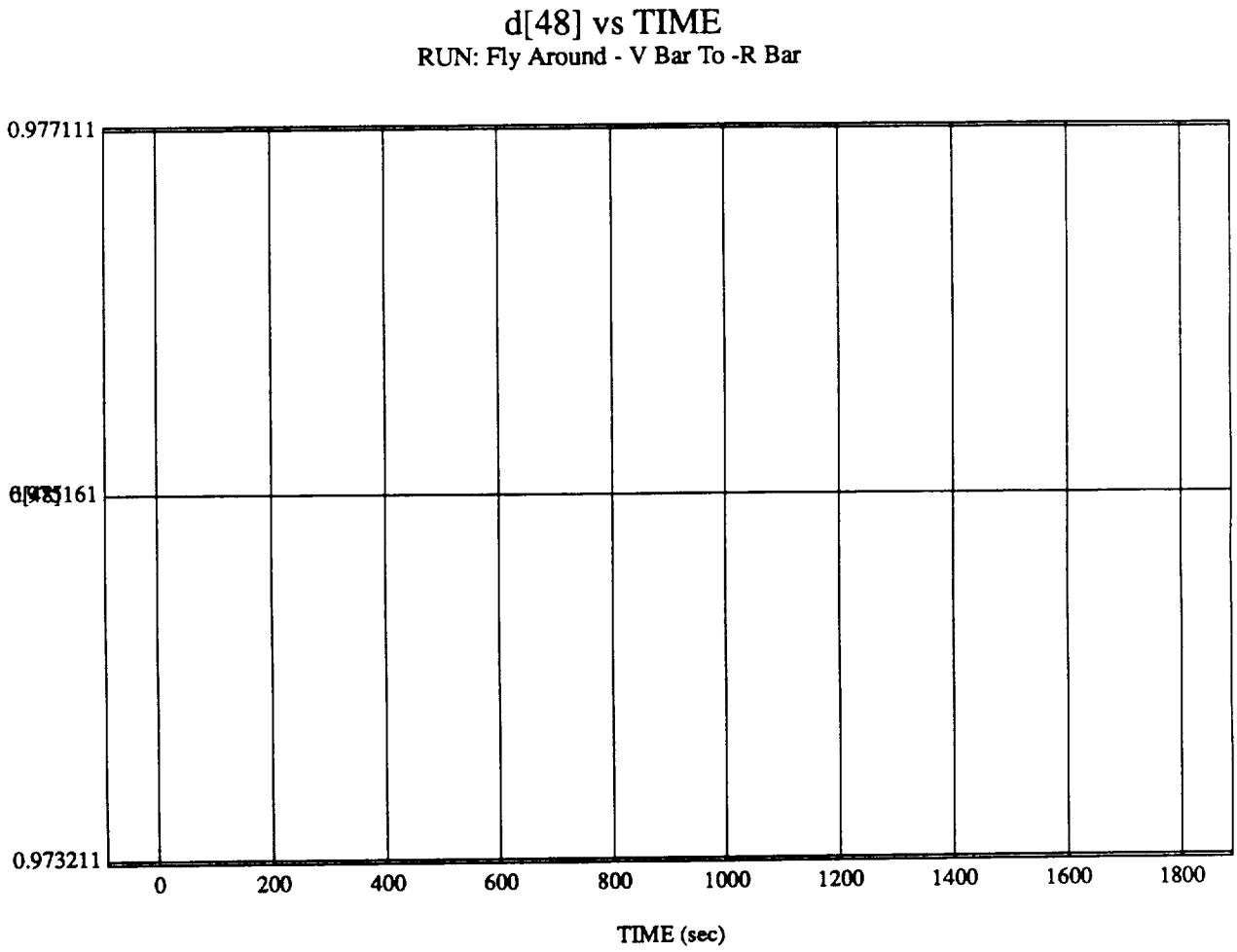
MODULE: ORBITER.lm_range
DATA SAMPLING FREQUENCY: 0.200 Hz



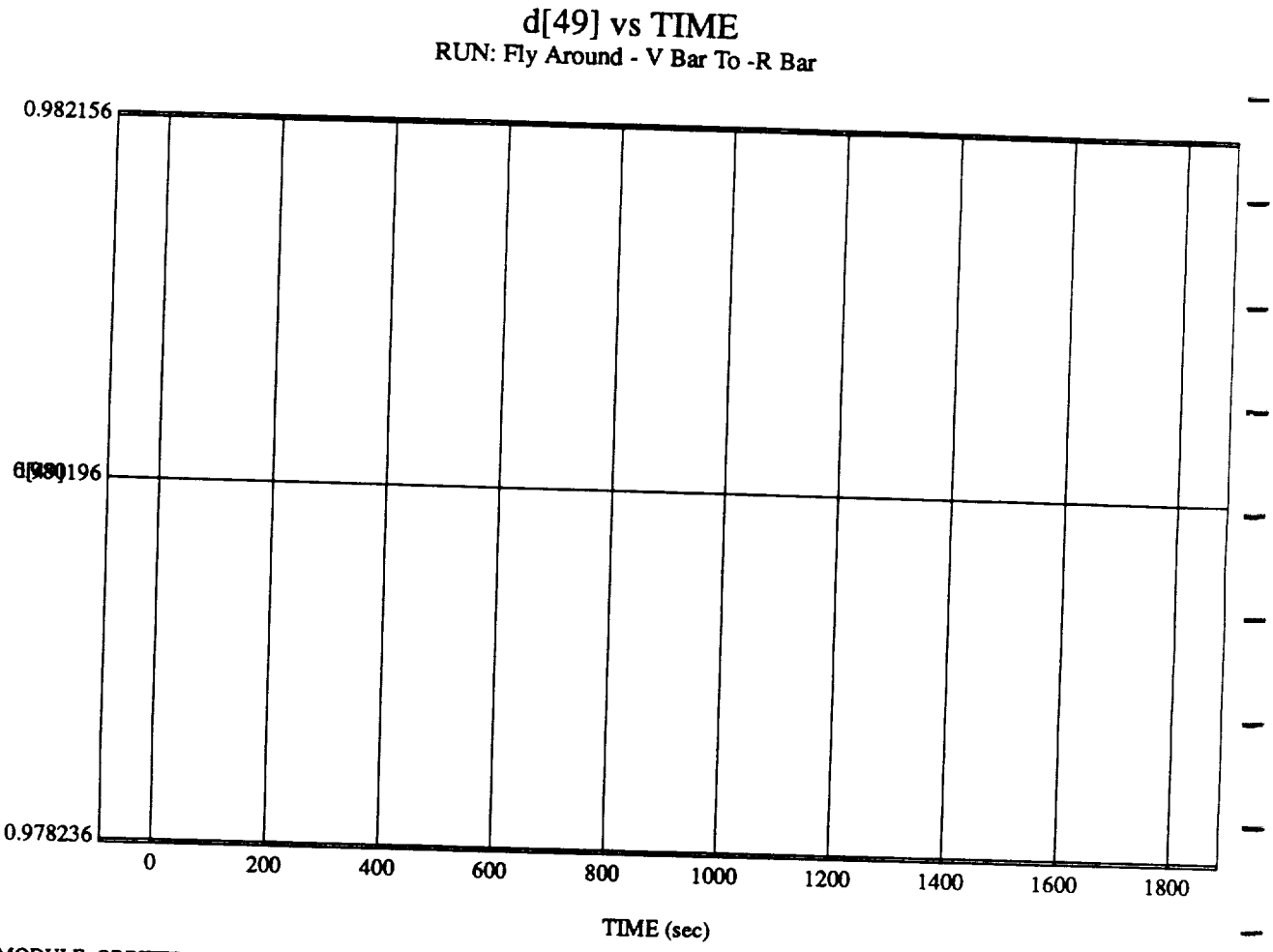
MODULE: ORBITER.lm_range
DATA SAMPLING FREQUENCY: 0.200 Hz



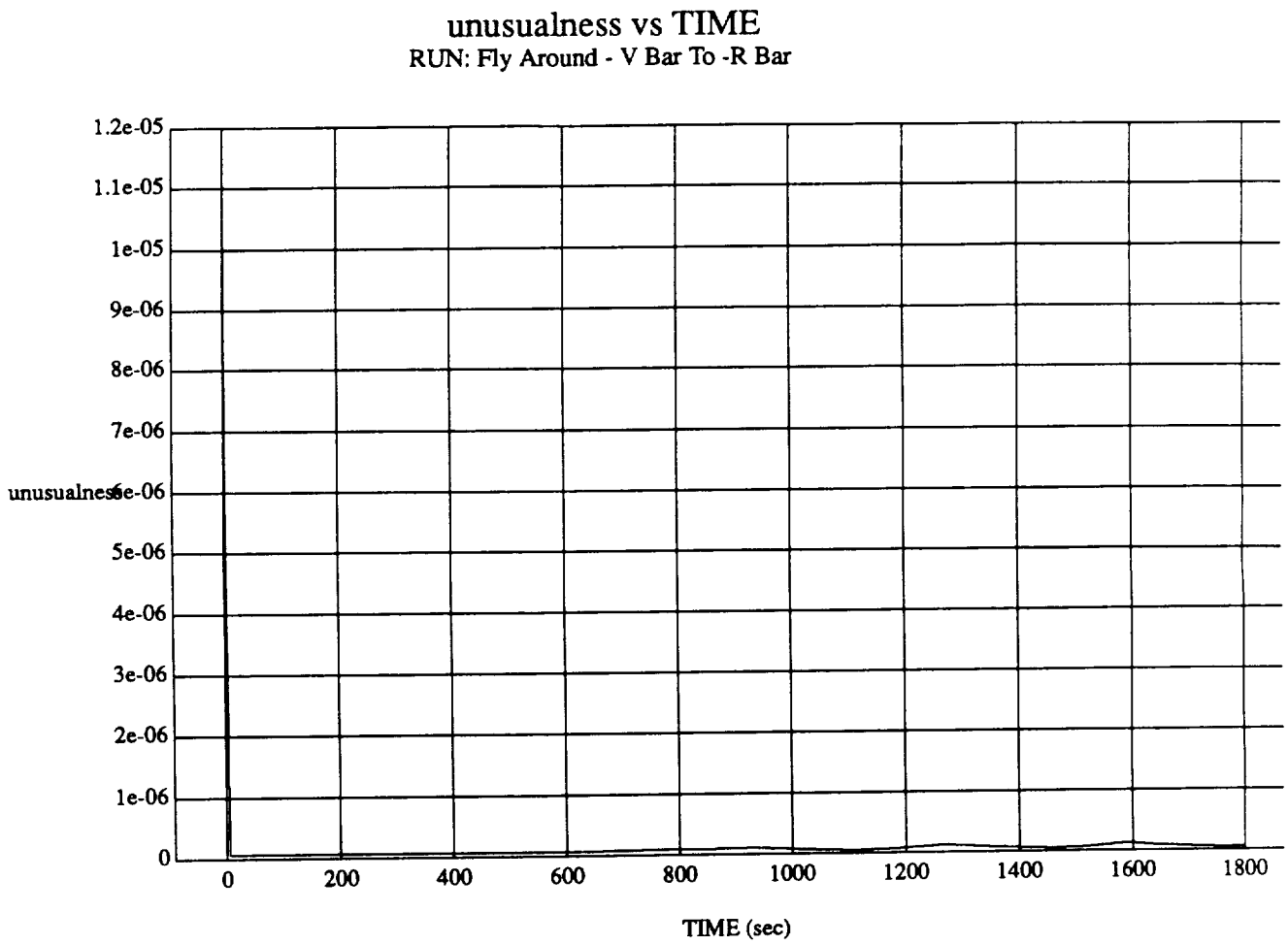
MODULE: ORBITER.lm_range
DATA SAMPLING FREQUENCY: 0.200 Hz



MODULE: ORBITER.lm_range
DATA SAMPLING FREQUENCY: 0.200 Hz



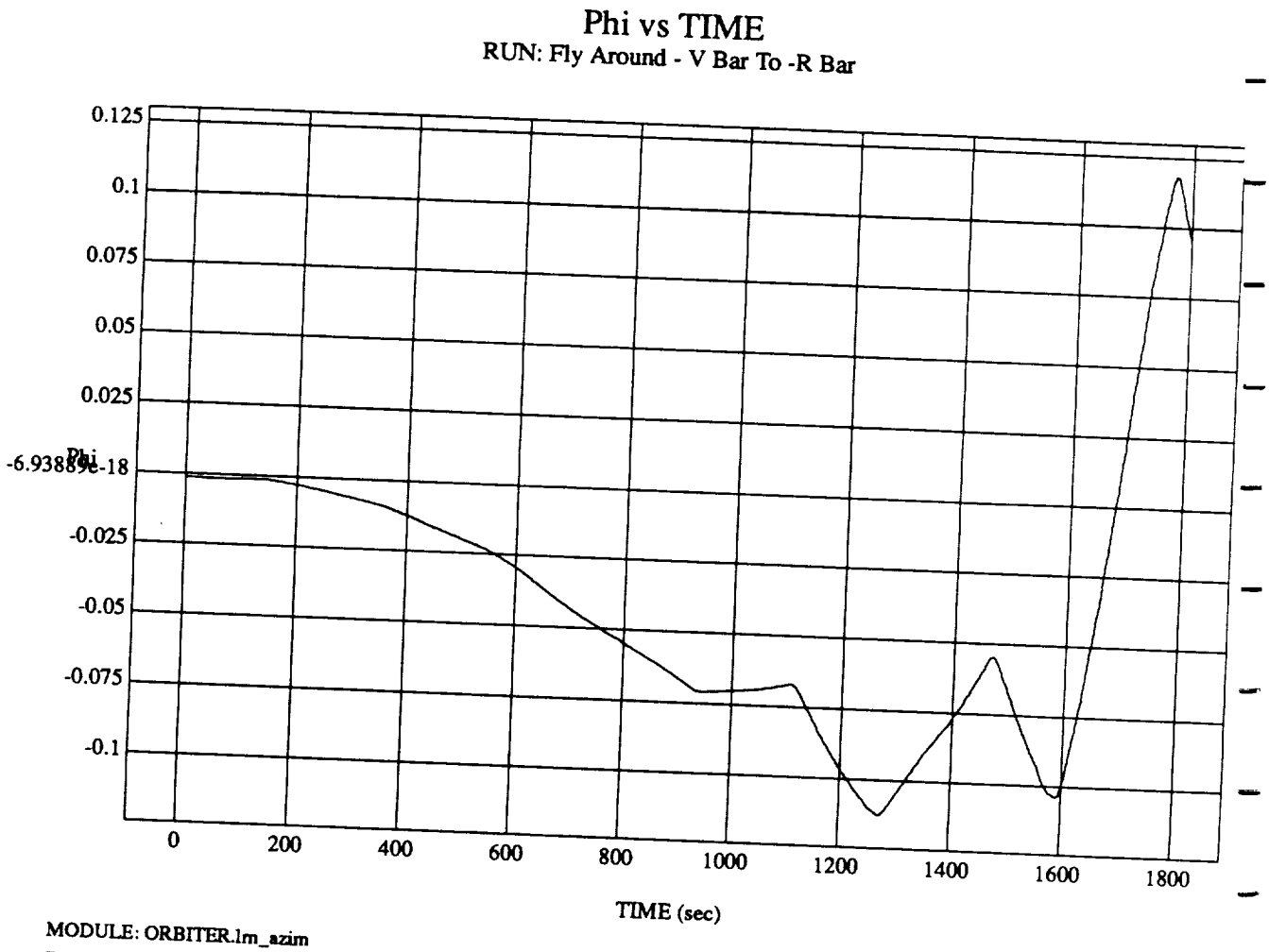
MODULE: ORBITER.lm_range
DATA SAMPLING FREQUENCY: 0.200 Hz



MODULE: ORBITER.lm_range
DATA SAMPLING FREQUENCY: 0.200 Hz

28

SIMULATION APPLICATION: ARIC Translational Controller Simulation

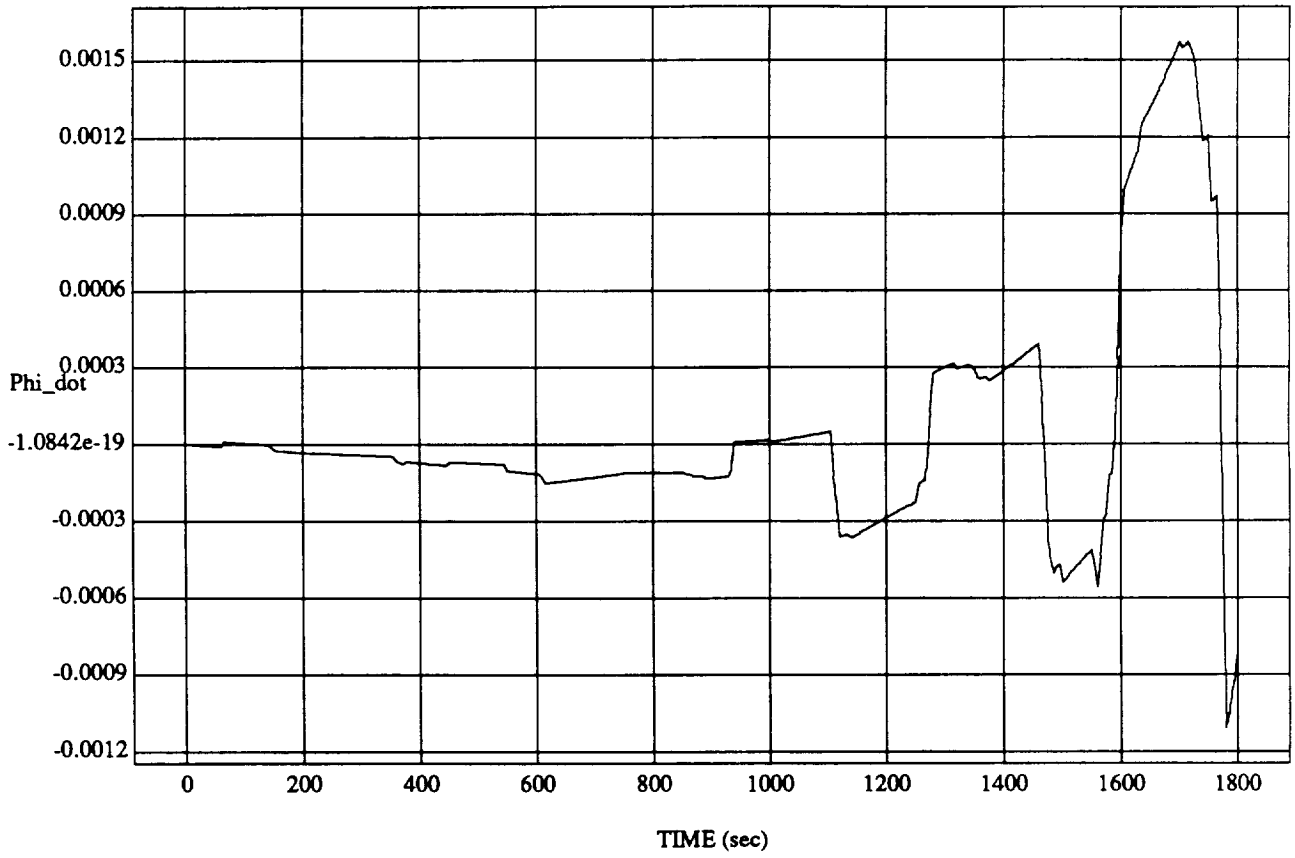


MODULE: ORBITER.lm_azim
DATA SAMPLING FREQUENCY: 0.200 Hz

SIMULATION APPLICATION: ARIC Translational Controller Simulation

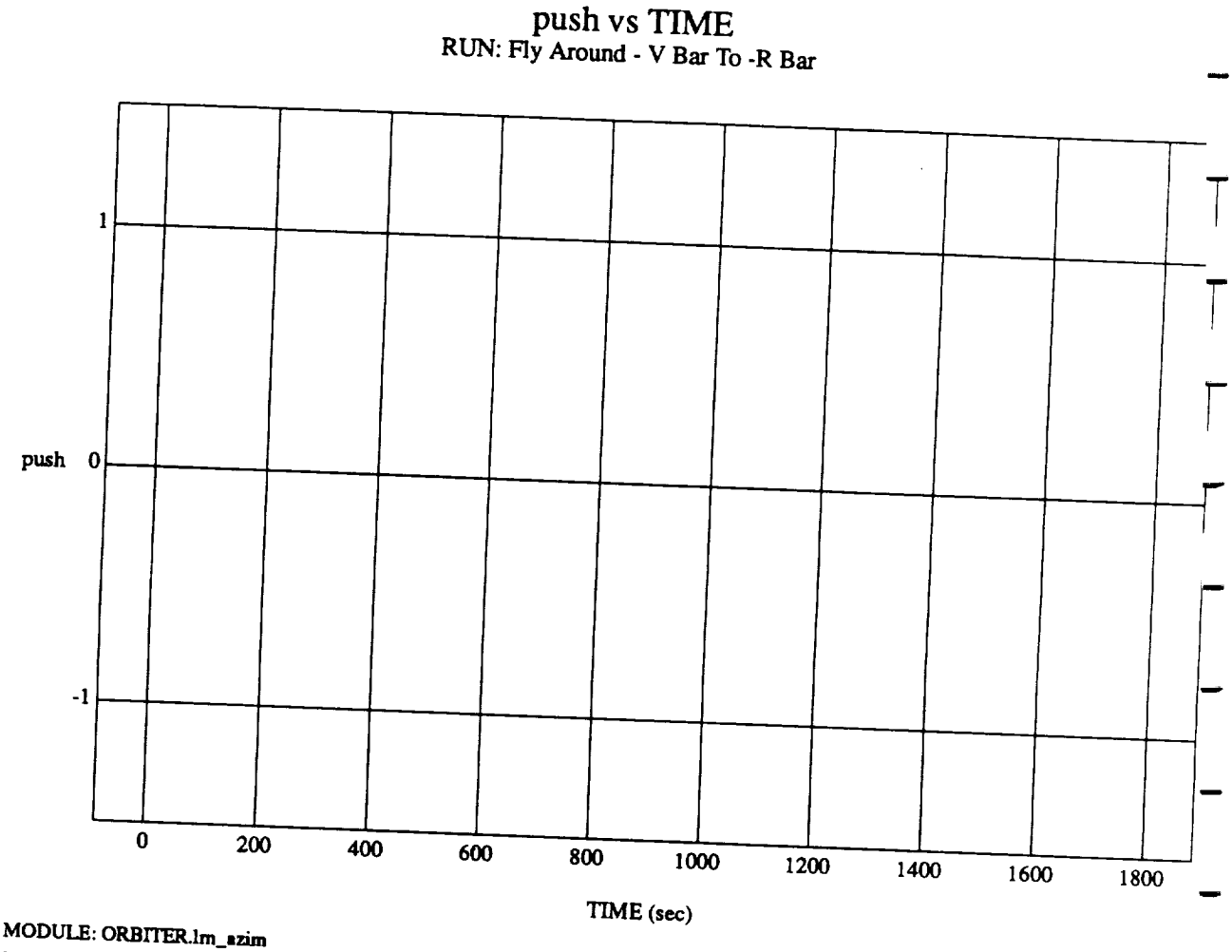
Phi_dot vs TIME

RUN: Fly Around - V Bar To -R Bar



MODULE: ORBITER.lm_azim
DATA SAMPLING FREQUENCY: 0.200 Hz

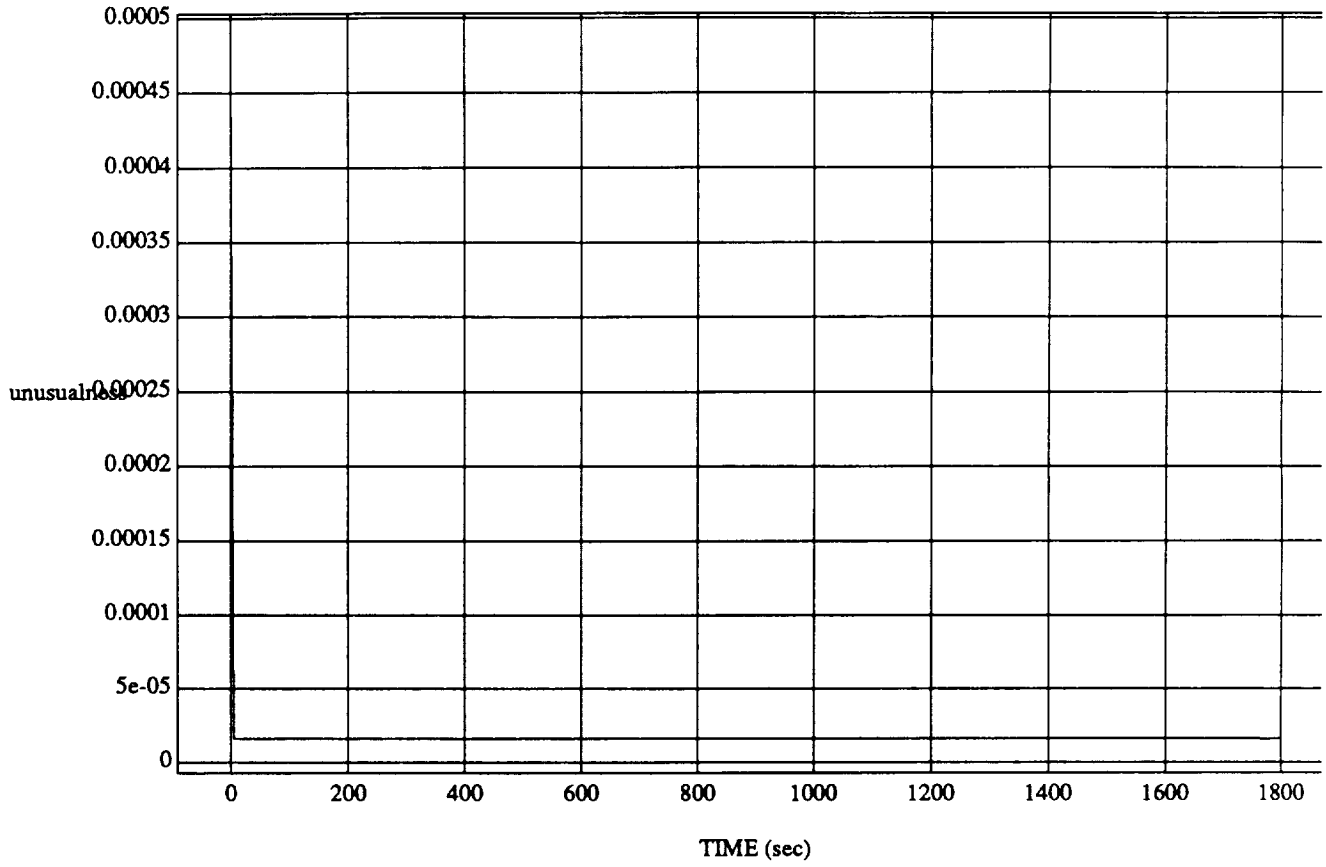
C-4



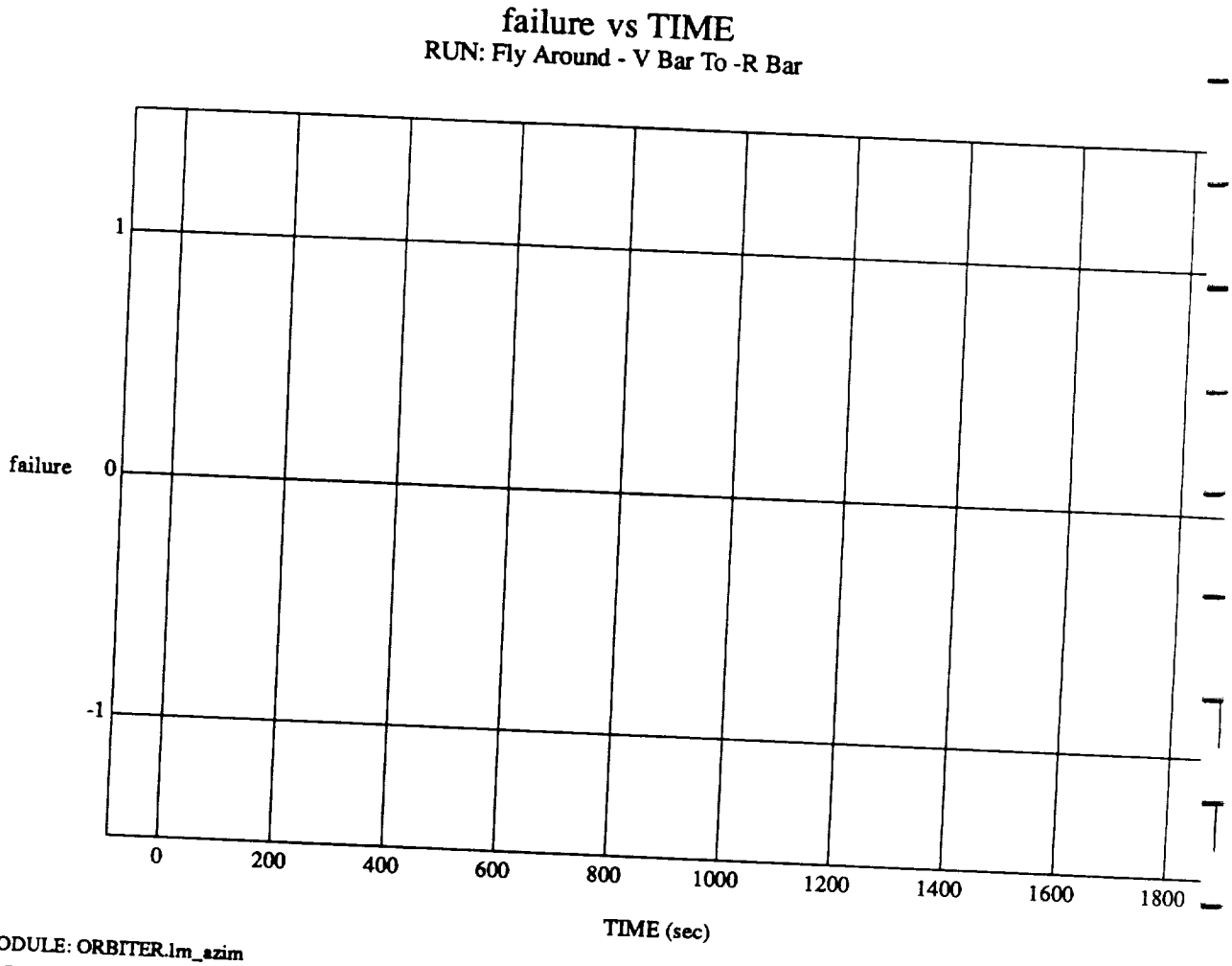
MODULE: ORBITER.lm_azim
DATA SAMPLING FREQUENCY: 0.200 Hz

unusualness vs TIME

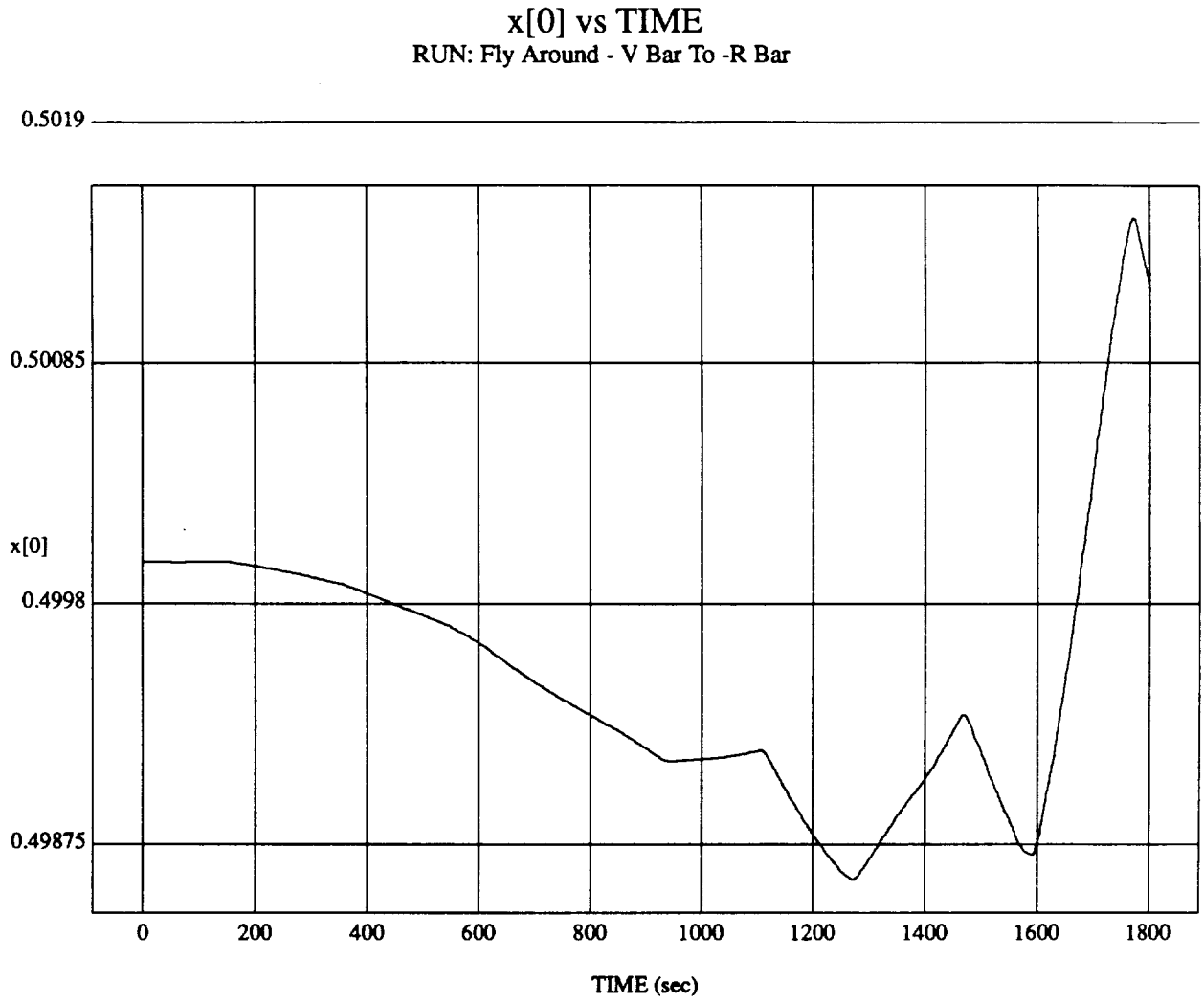
RUN: Fly Around - V Bar To -R Bar



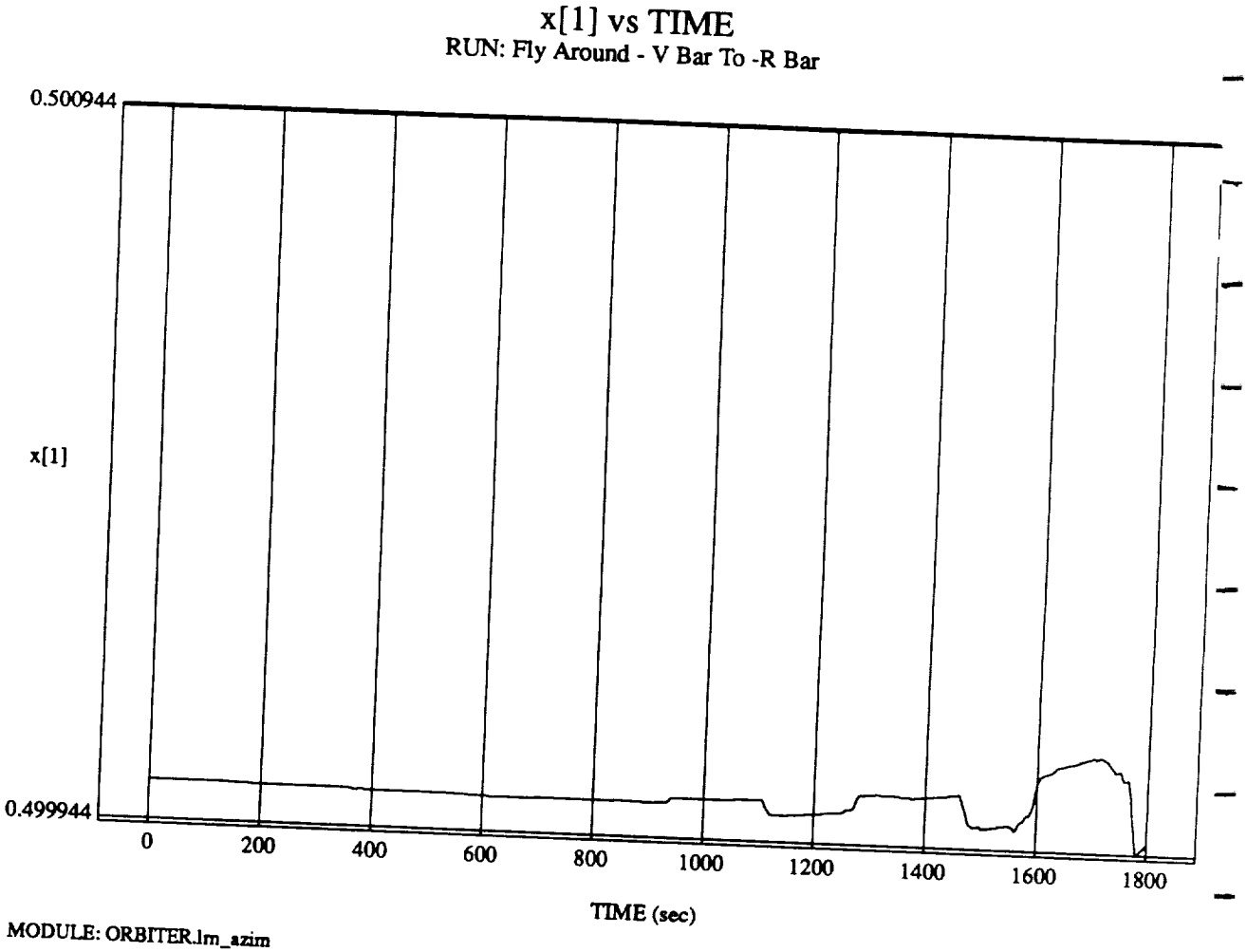
MODULE: ORBITER.lm_azim
DATA SAMPLING FREQUENCY: 0.200 Hz



MODULE: ORBITER.lm_azim
DATA SAMPLING FREQUENCY: 0.200 Hz

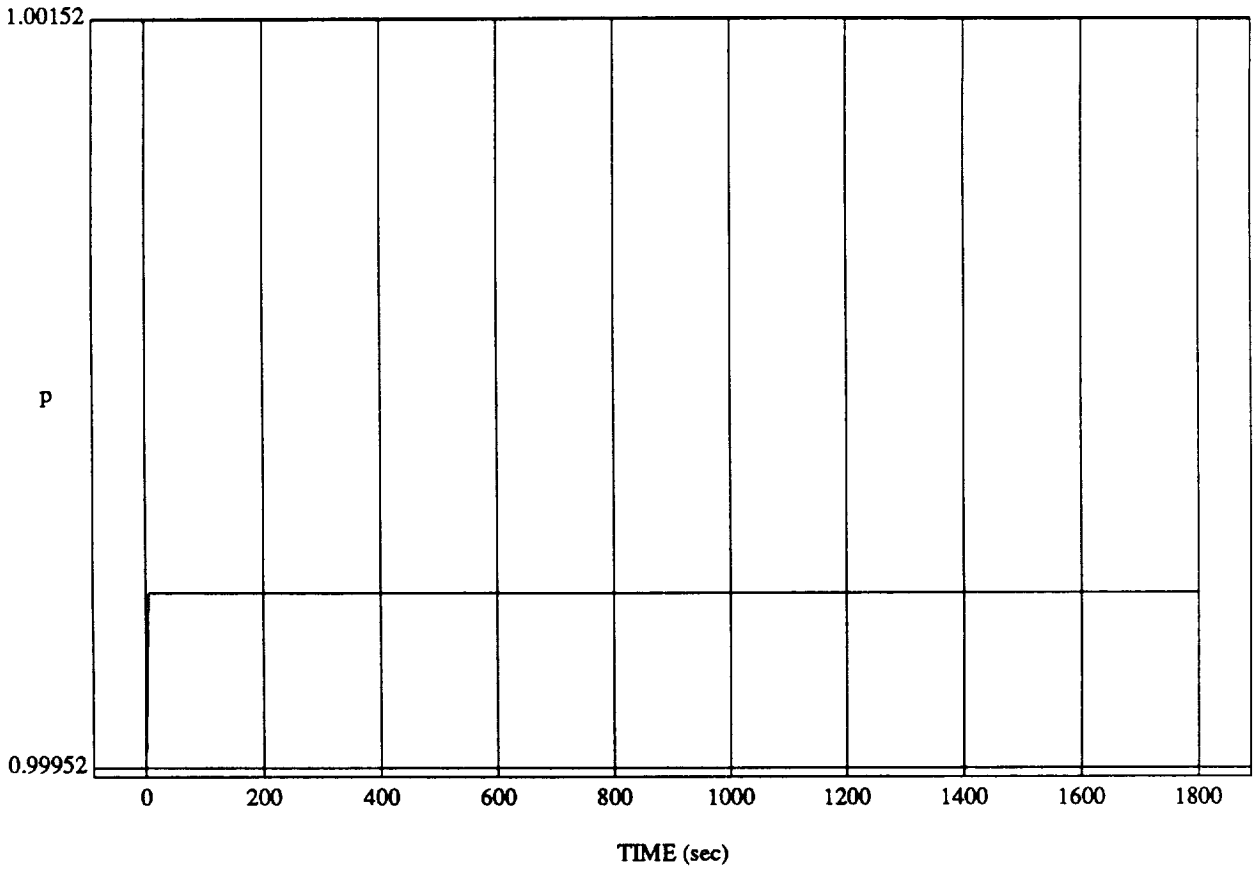


MODULE: ORBITER.lm_azim
DATA SAMPLING FREQUENCY: 0.200 Hz



MODULE: ORBITER.lm_azim
DATA SAMPLING FREQUENCY: 0.200 Hz

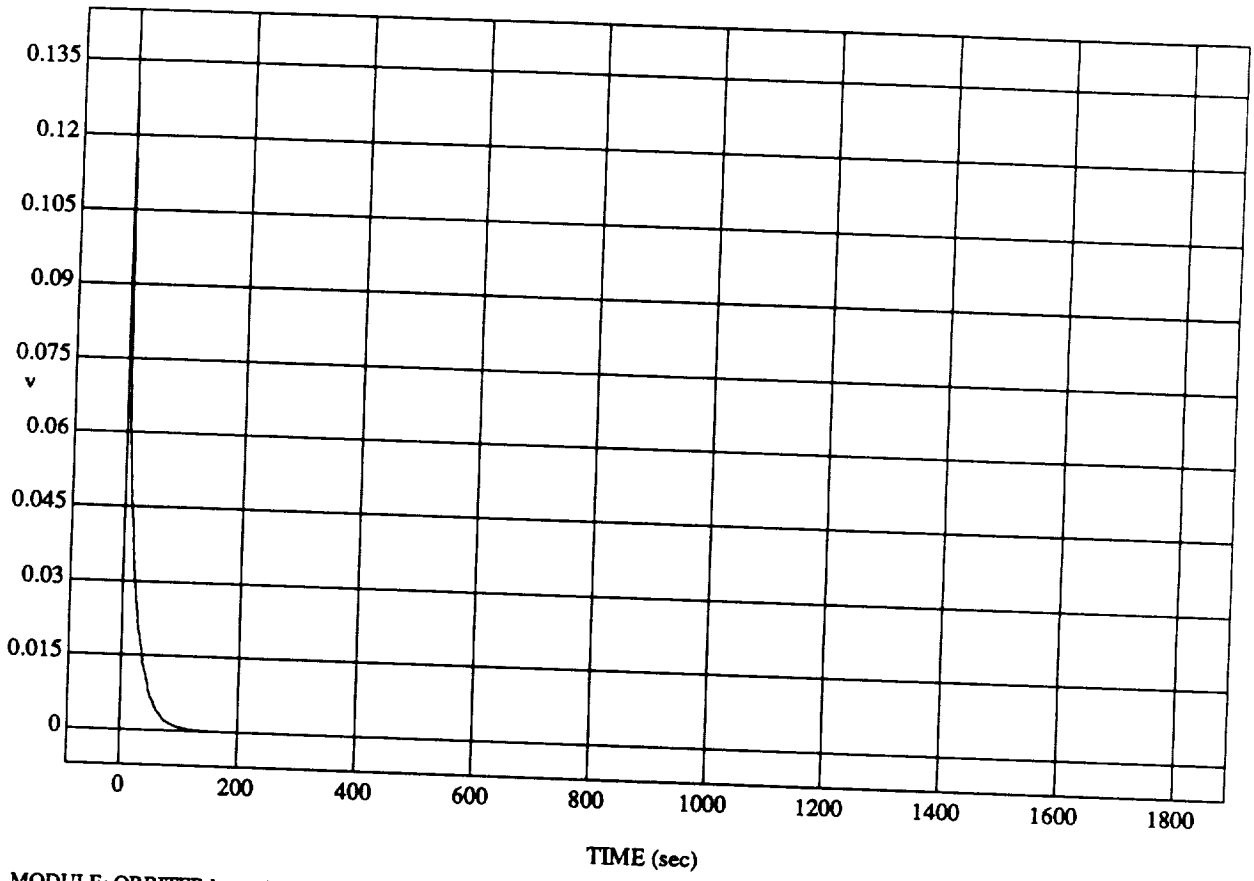
p vs TIME
RUN: Fly Around - V Bar To -R Bar



MODULE: ORBITER.lm_azim
DATA SAMPLING FREQUENCY: 0.200 Hz

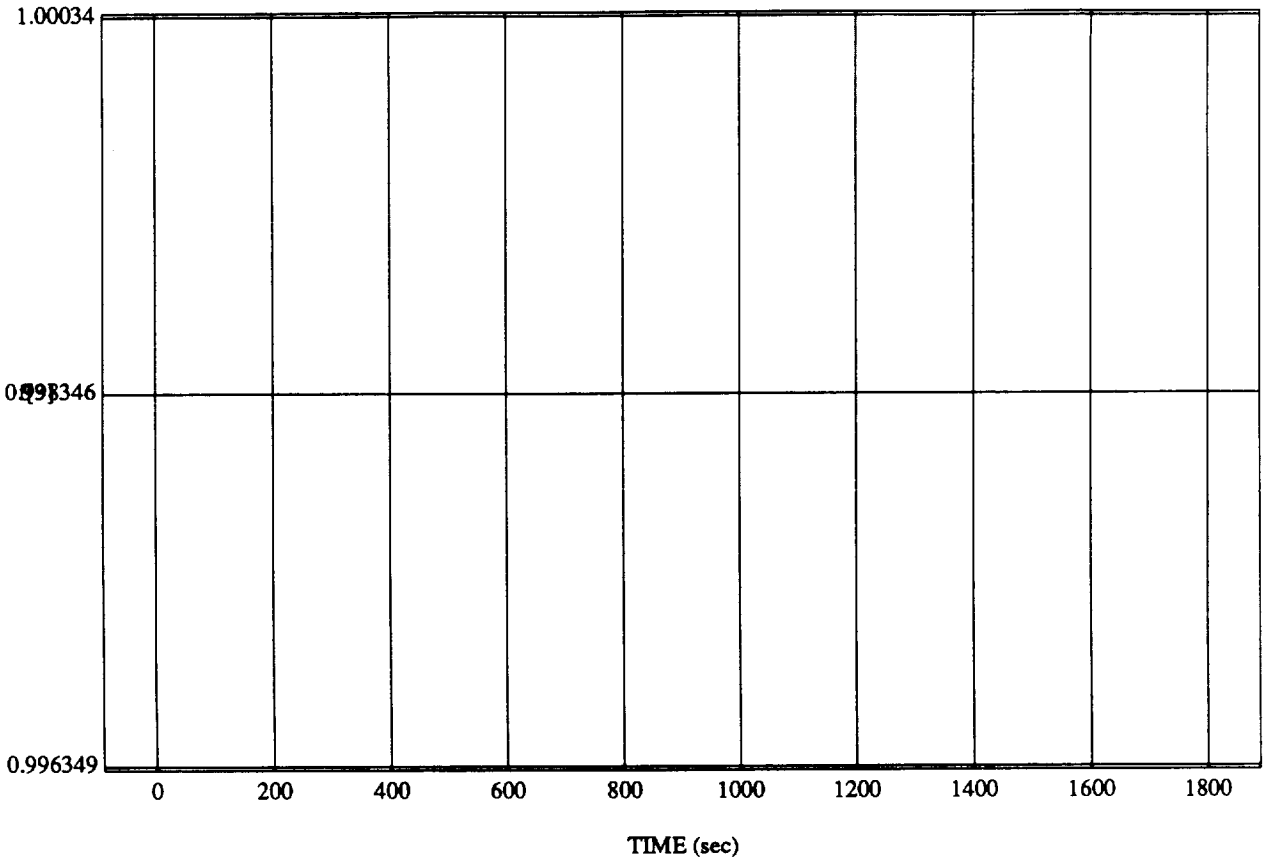
V vs TIME

RUN: Fly Around - V Bar To -R Bar

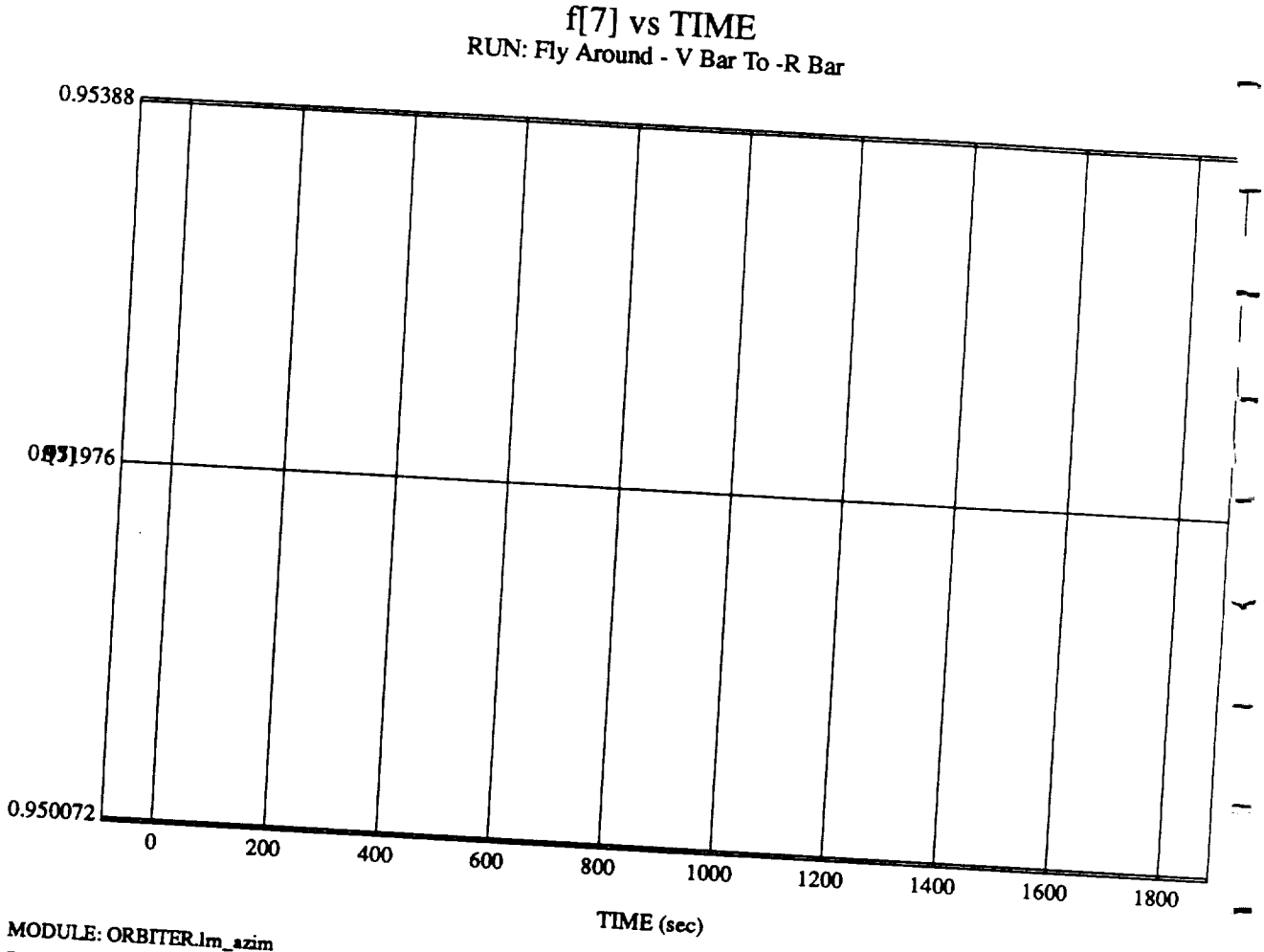


MODULE: ORBITER.lm_azim
DATA SAMPLING FREQUENCY: 0.200 Hz

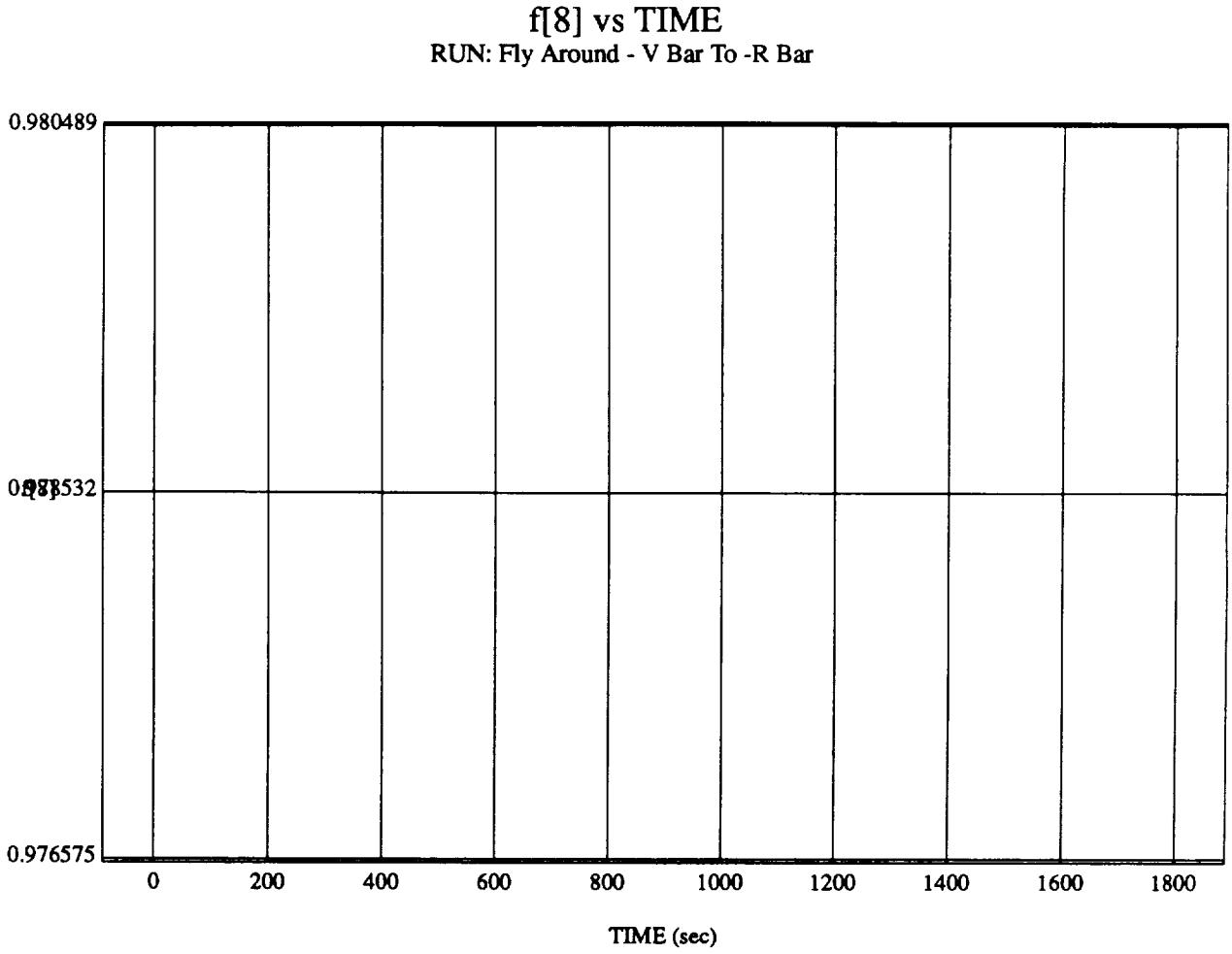
f[3] vs TIME
RUN: Fly Around - V Bar To -R Bar



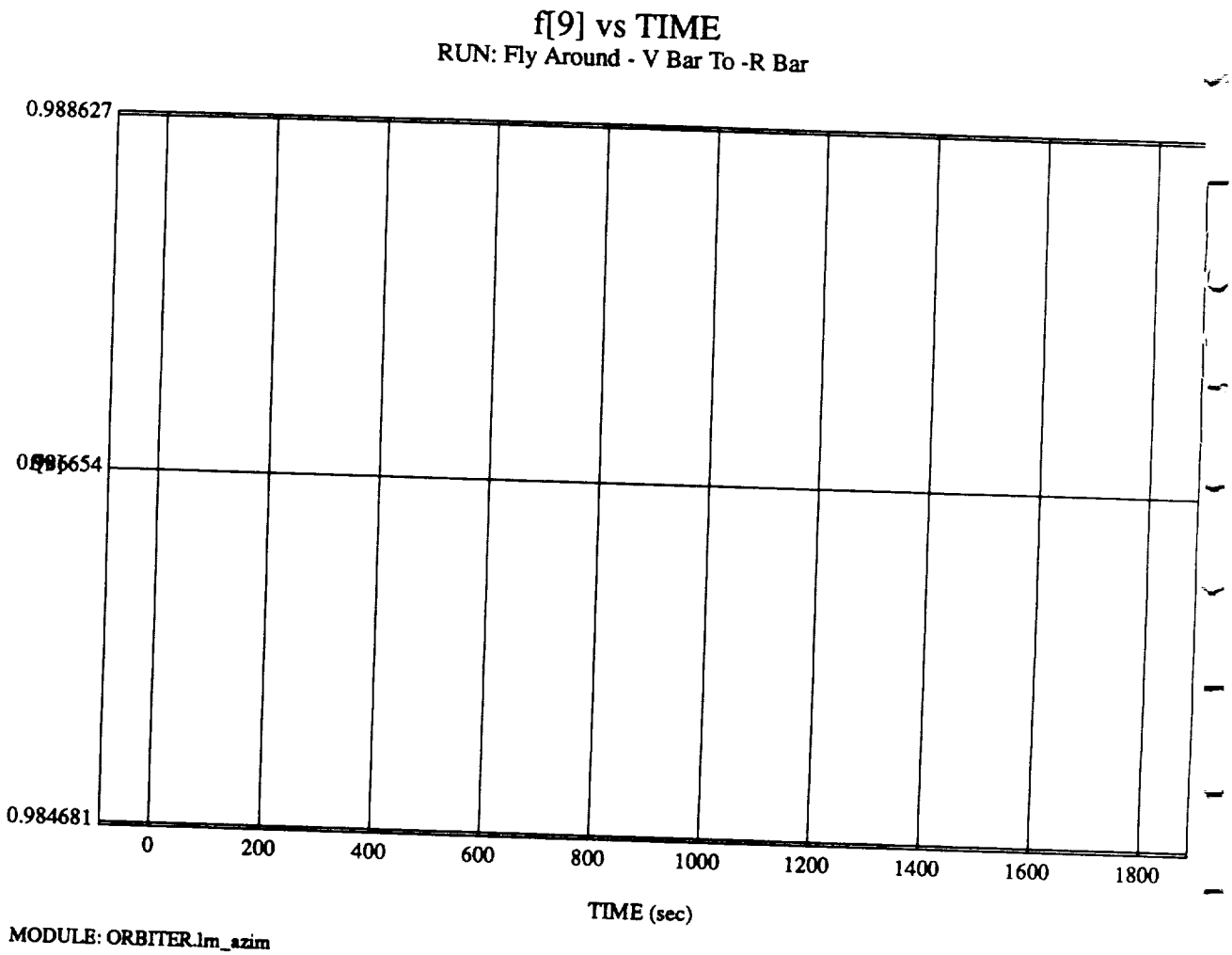
MODULE: ORBITER.lm_azim
DATA SAMPLING FREQUENCY: 0.200 Hz



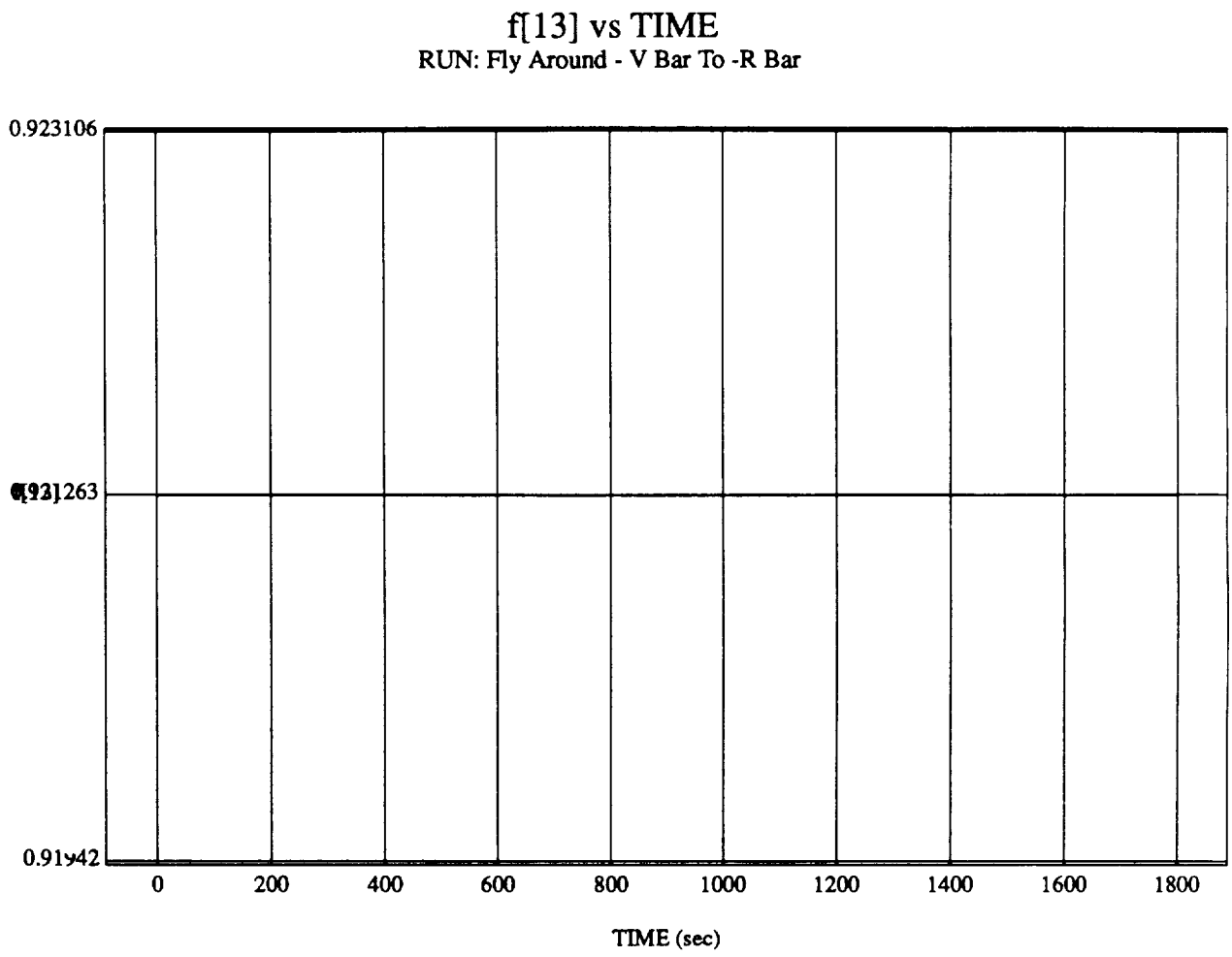
MODULE: ORBITER.lm_azim
DATA SAMPLING FREQUENCY: 0.200 Hz



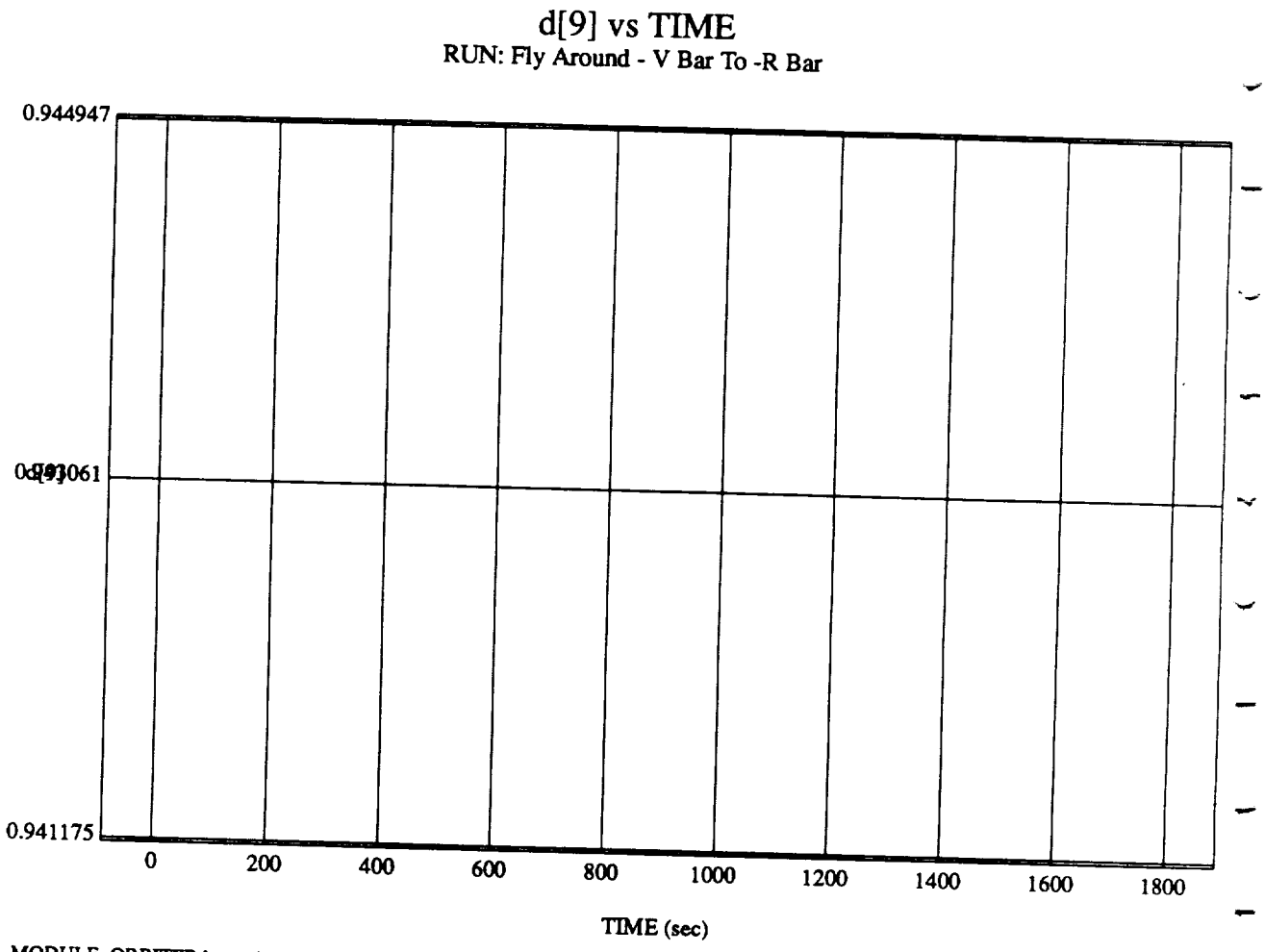
MODULE: ORBITER.lm_azim
DATA SAMPLING FREQUENCY: 0.200 Hz



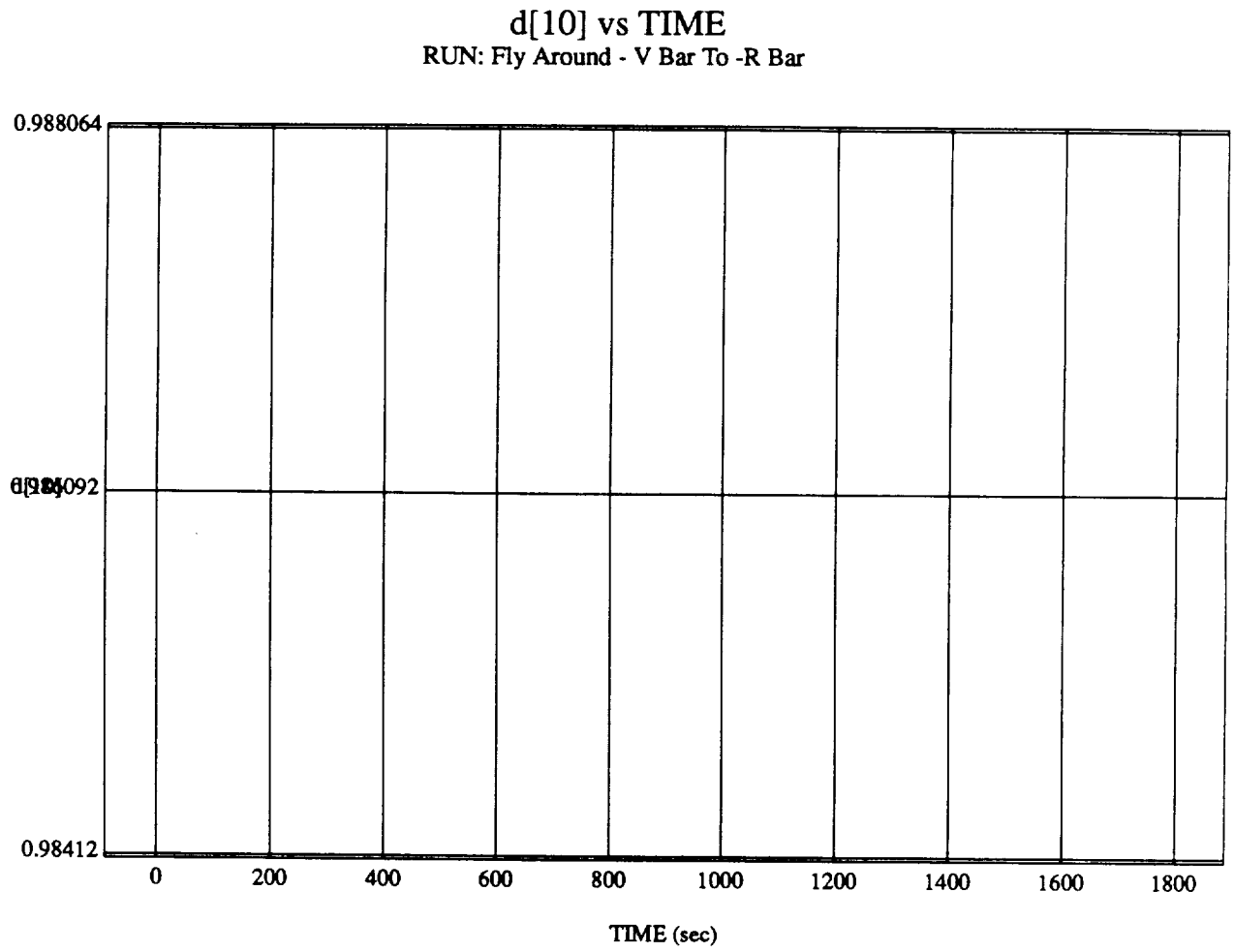
MODULE: ORBITER.lm_azim
DATA SAMPLING FREQUENCY: 0.200 Hz



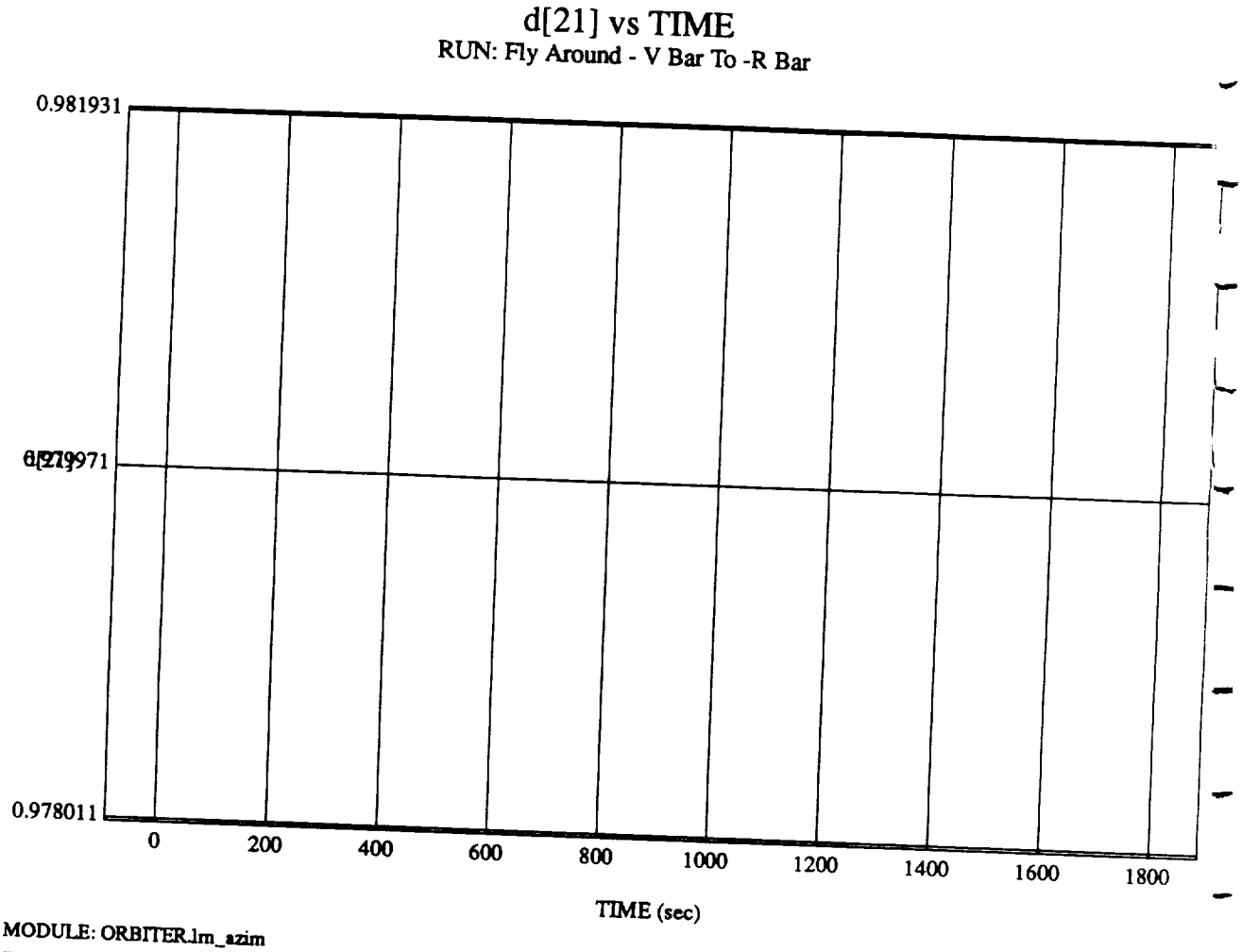
MODULE: ORBITER.lm_azim
DATA SAMPLING FREQUENCY: 0.200 Hz



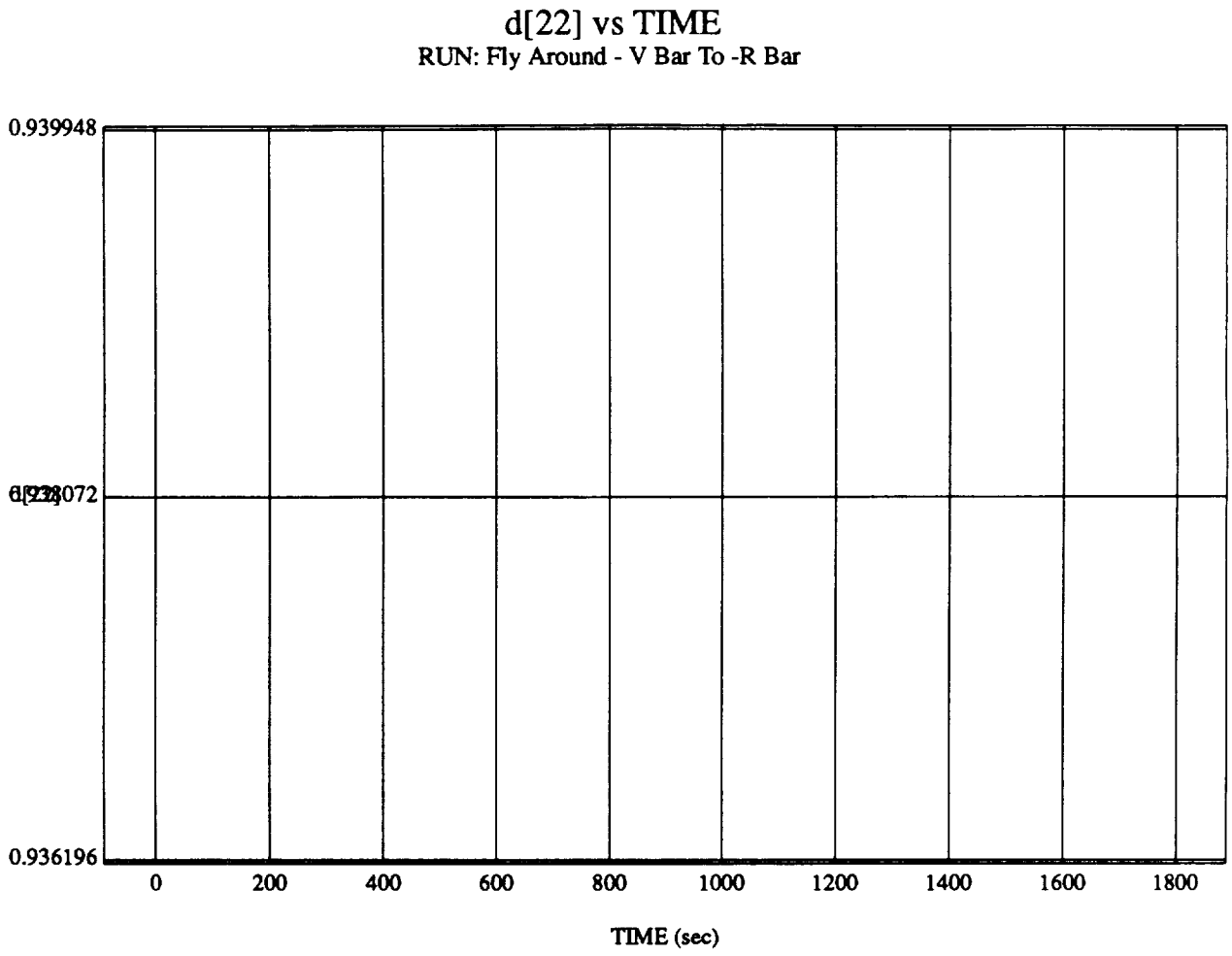
MODULE: ORBITER.lm_azim
DATA SAMPLING FREQUENCY: 0.200 Hz



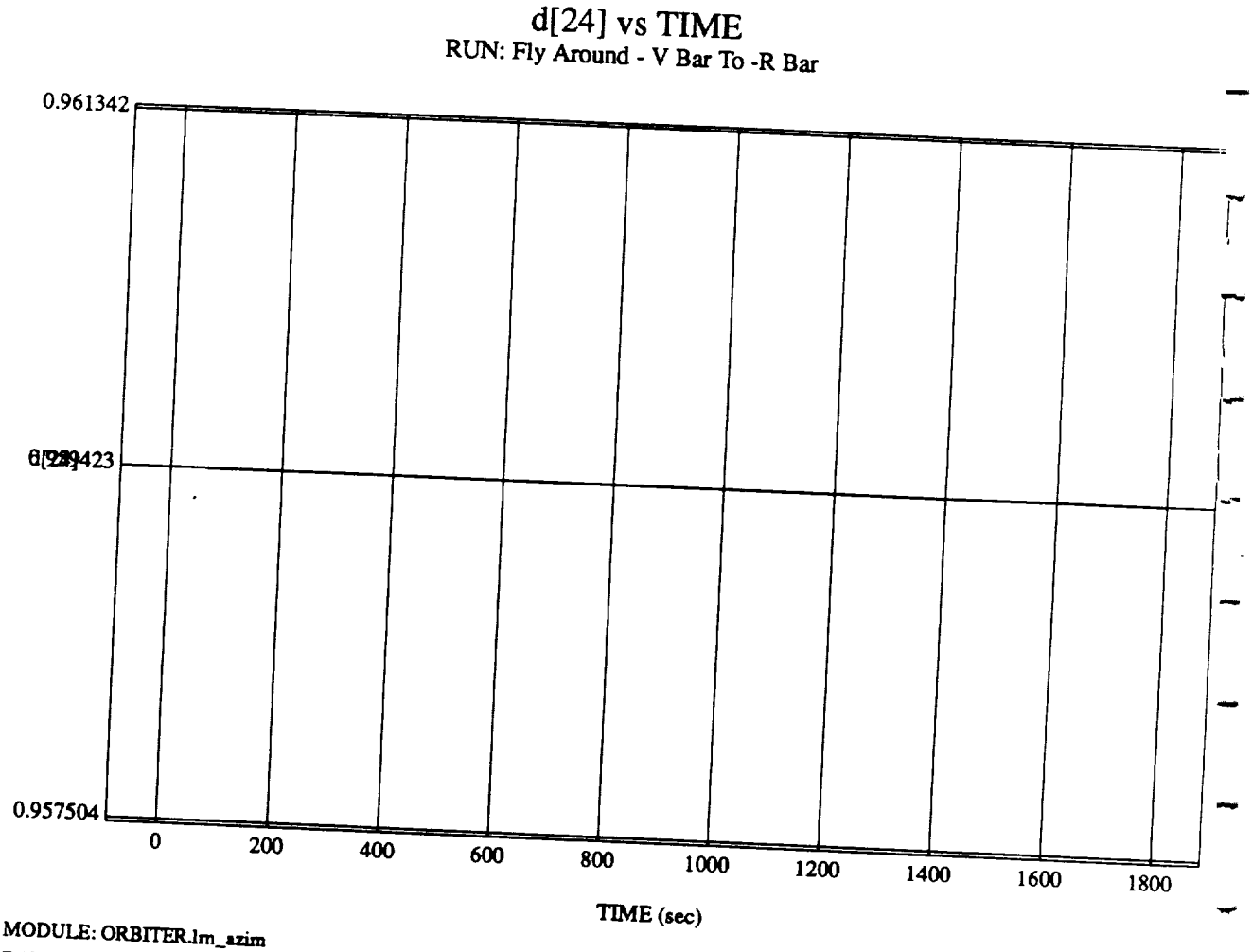
MODULE: ORBITER.lm_azim
DATA SAMPLING FREQUENCY: 0.200 Hz



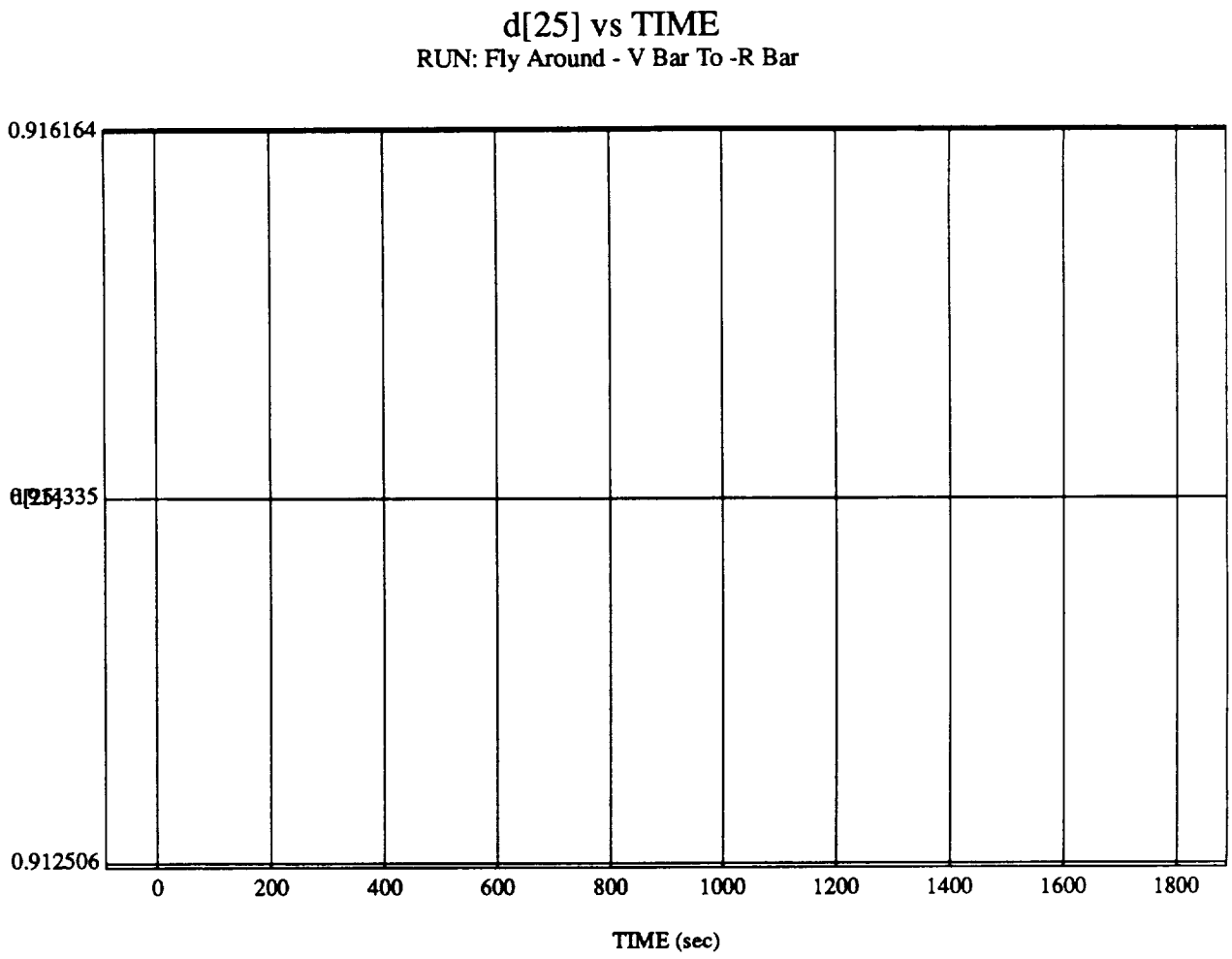
MODULE: ORBITER.lm_azim
DATA SAMPLING FREQUENCY: 0.200 Hz



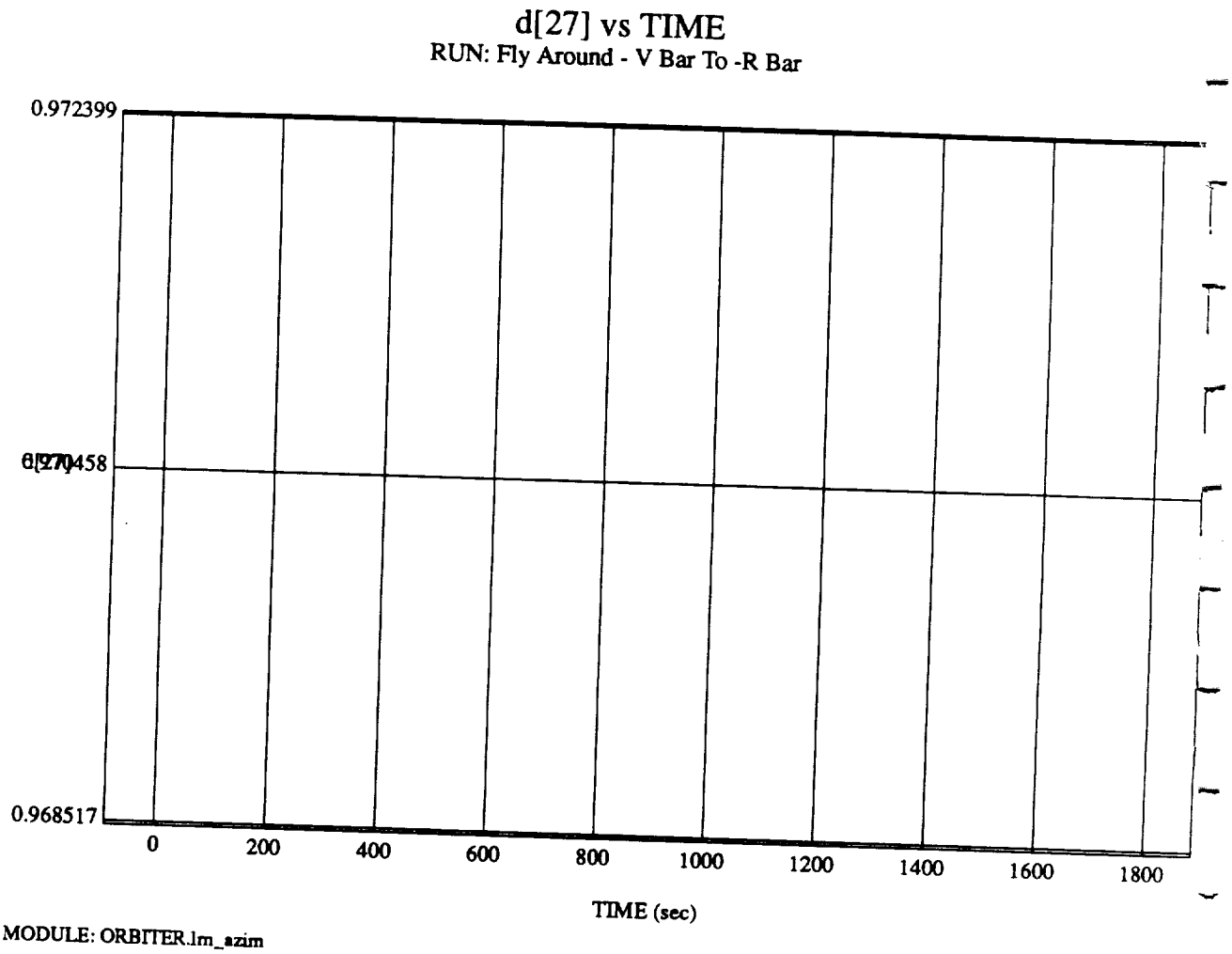
MODULE: ORBITER.lm_azim
DATA SAMPLING FREQUENCY: 0.200 Hz



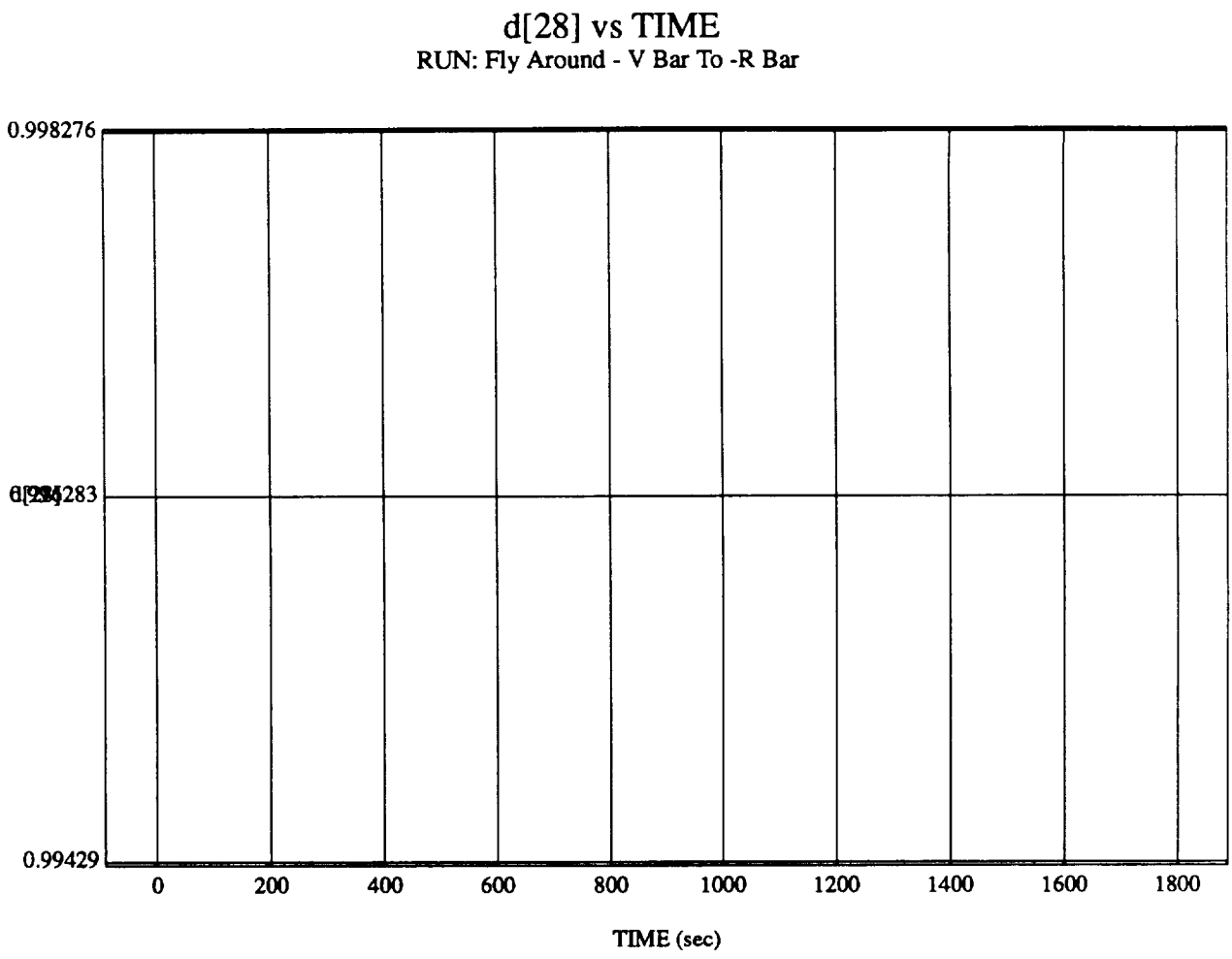
MODULE: ORBITER.lm_azim
DATA SAMPLING FREQUENCY: 0.200 Hz



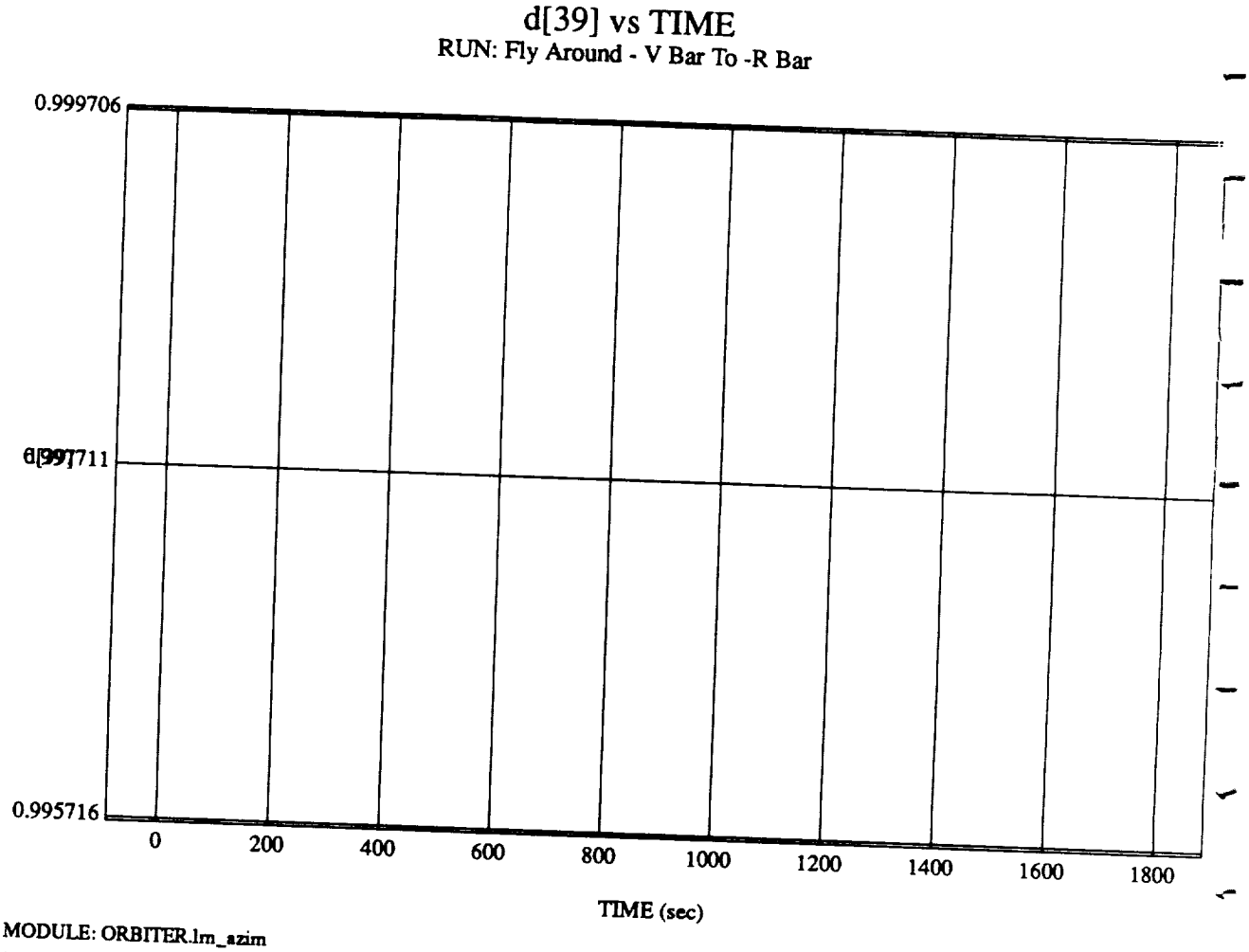
MODULE: ORBITER.lm_azim
DATA SAMPLING FREQUENCY: 0.200 Hz



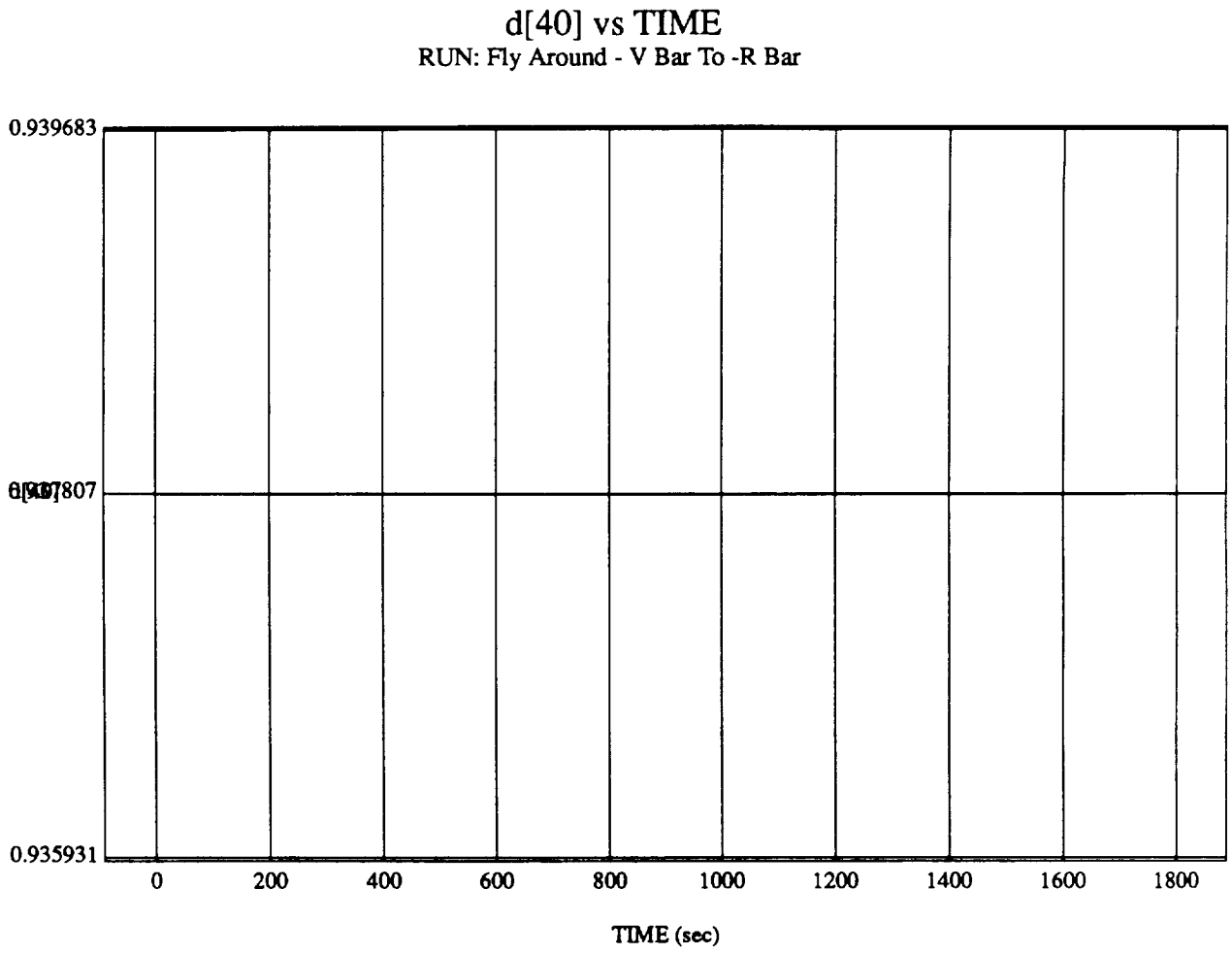
MODULE: ORBITER.lm_azim
DATA SAMPLING FREQUENCY: 0.200 Hz



MODULE: ORBITER.lm_azim
DATA SAMPLING FREQUENCY: 0.200 Hz



MODULE: ORBITER.lm_azim
DATA SAMPLING FREQUENCY: 0.200 Hz



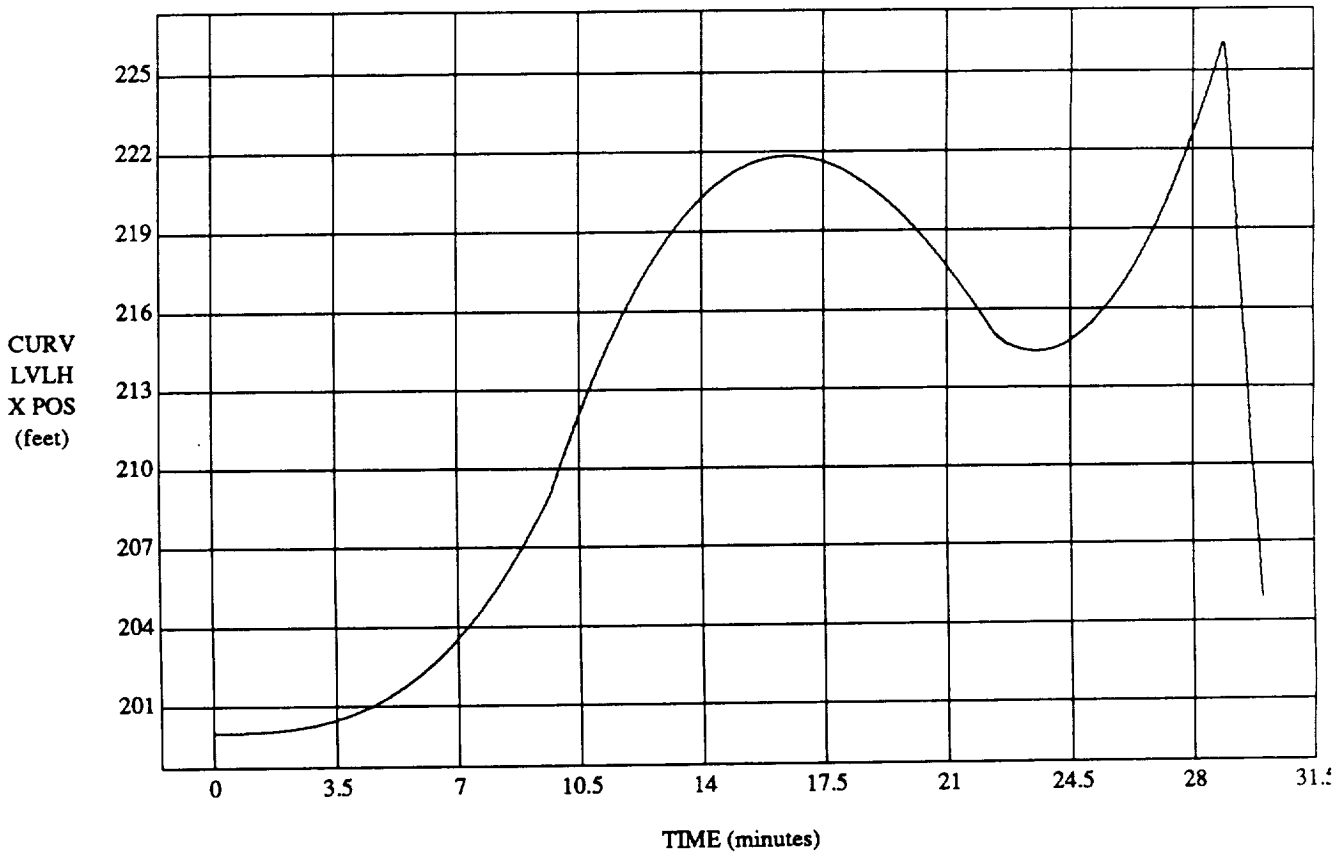
MODULE: ORBITER.lm_azim
DATA SAMPLING FREQUENCY: 0.200 Hz

B4. Station-Keeping for 30 minutes at 50 feet distance on V-bar

SIMULATION APPLICATION: ARIC Translational Controller Simulation

TARGET CENTERED ROTATING CURVILINEAR LVLH X POS vs TIME

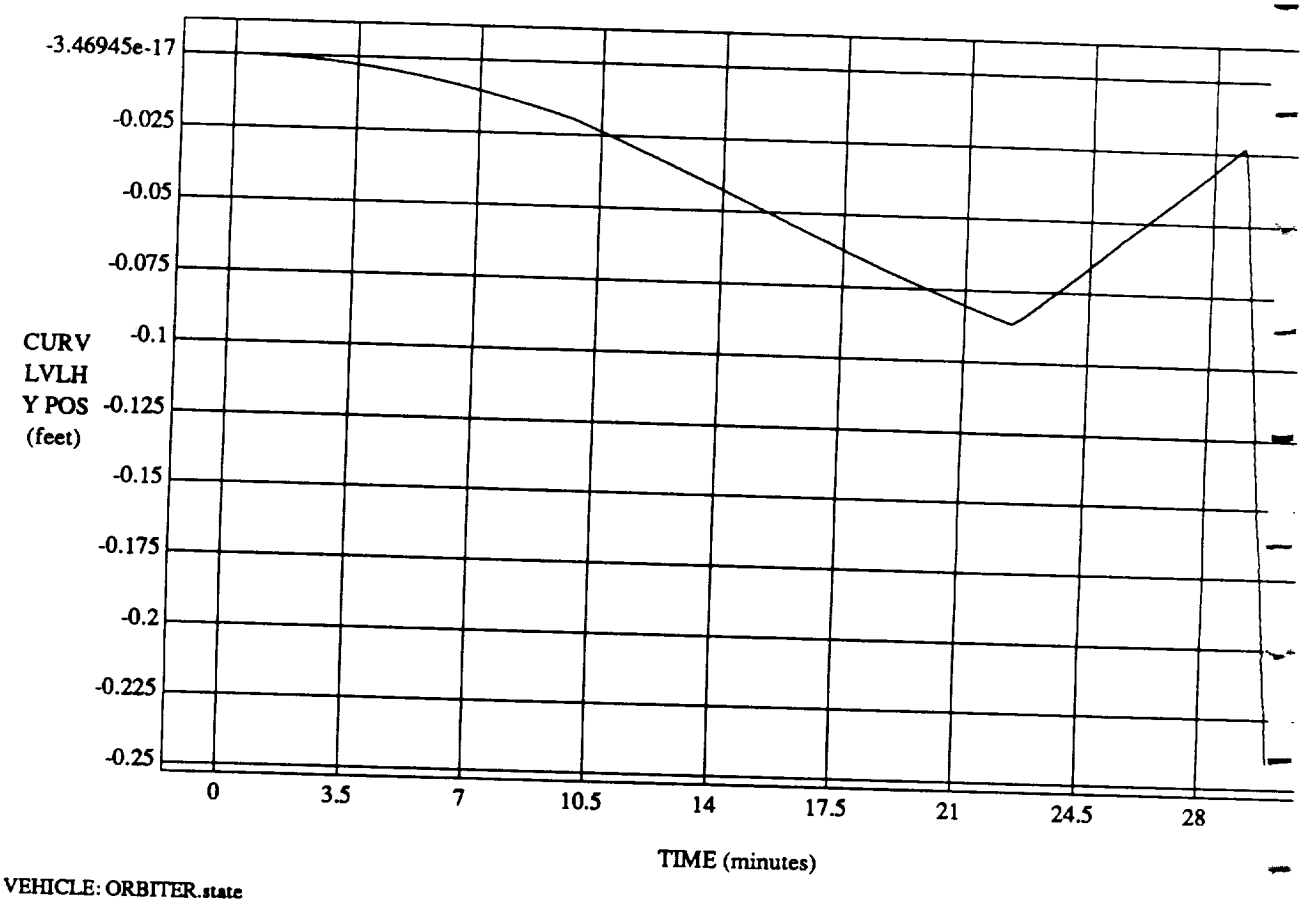
RUN: Station Keep At 200 Feet



VEHICLE: ORBITER.state
TARGET VEHICLE: SOLMAX.state
DATA SAMPLING FREQUENCY: 0.521 Hz

TARGET CENTERED ROTATING CURVILINEAR LVLH Y POS vs TIME

RUN: Station Keep At 200 Feet



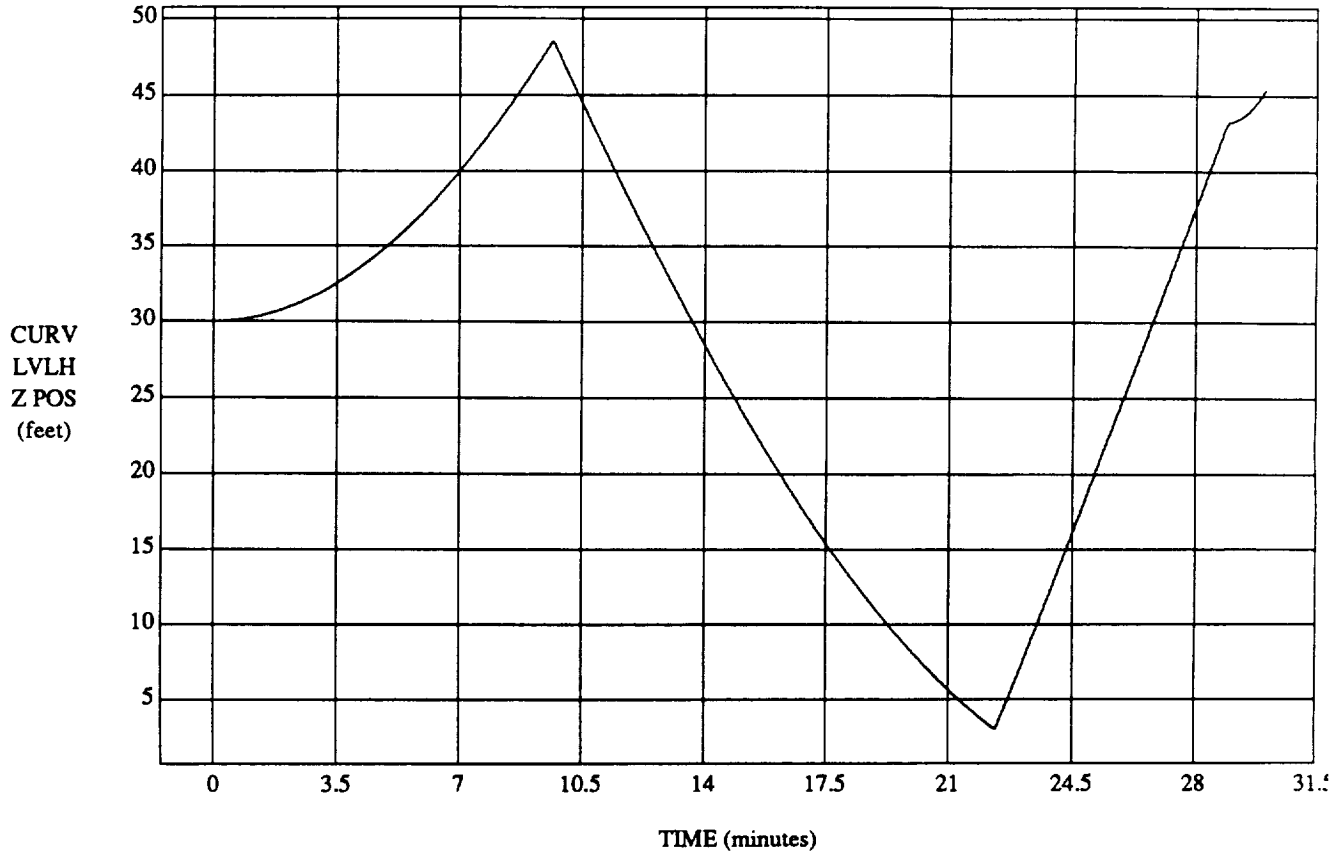
VEHICLE: ORBITER.state
TARGET VEHICLE: SOLMAX.state
DATA SAMPLING FREQUENCY: 0.521 Hz

315

SIMULATION APPLICATION: ARIC Translational Controller Simulation

TARGET CENTERED ROTATING CURVILINEAR LVLH Z POS vs TIME

RUN: Station Keep At 200 Feet

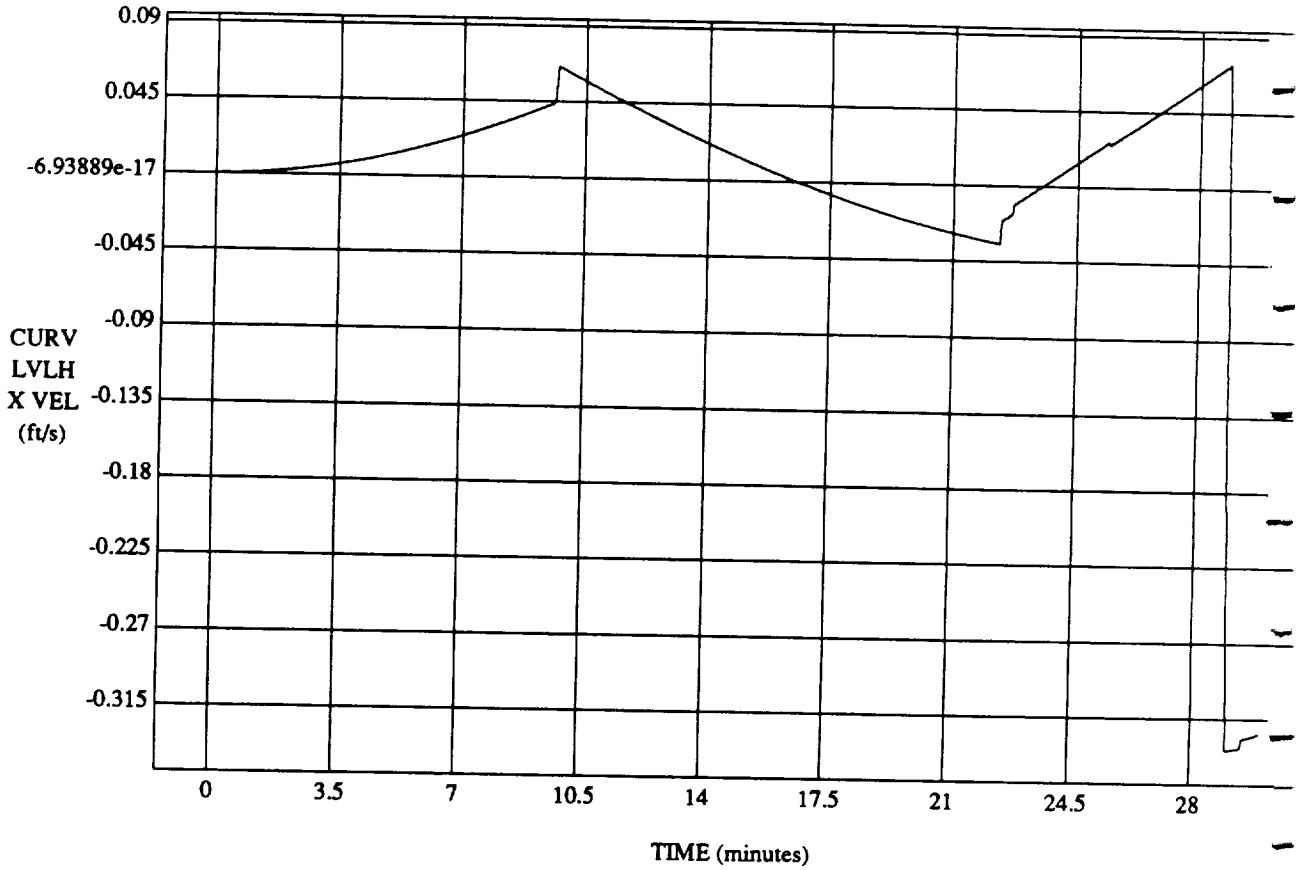


VEHICLE: ORBITER.state
TARGET VEHICLE: SOLMAX.state
DATA SAMPLING FREQUENCY: 0.521 Hz

3/10

TARGET CENTERED ROTATING CURVILINEAR LVLH X VEL vs TIME

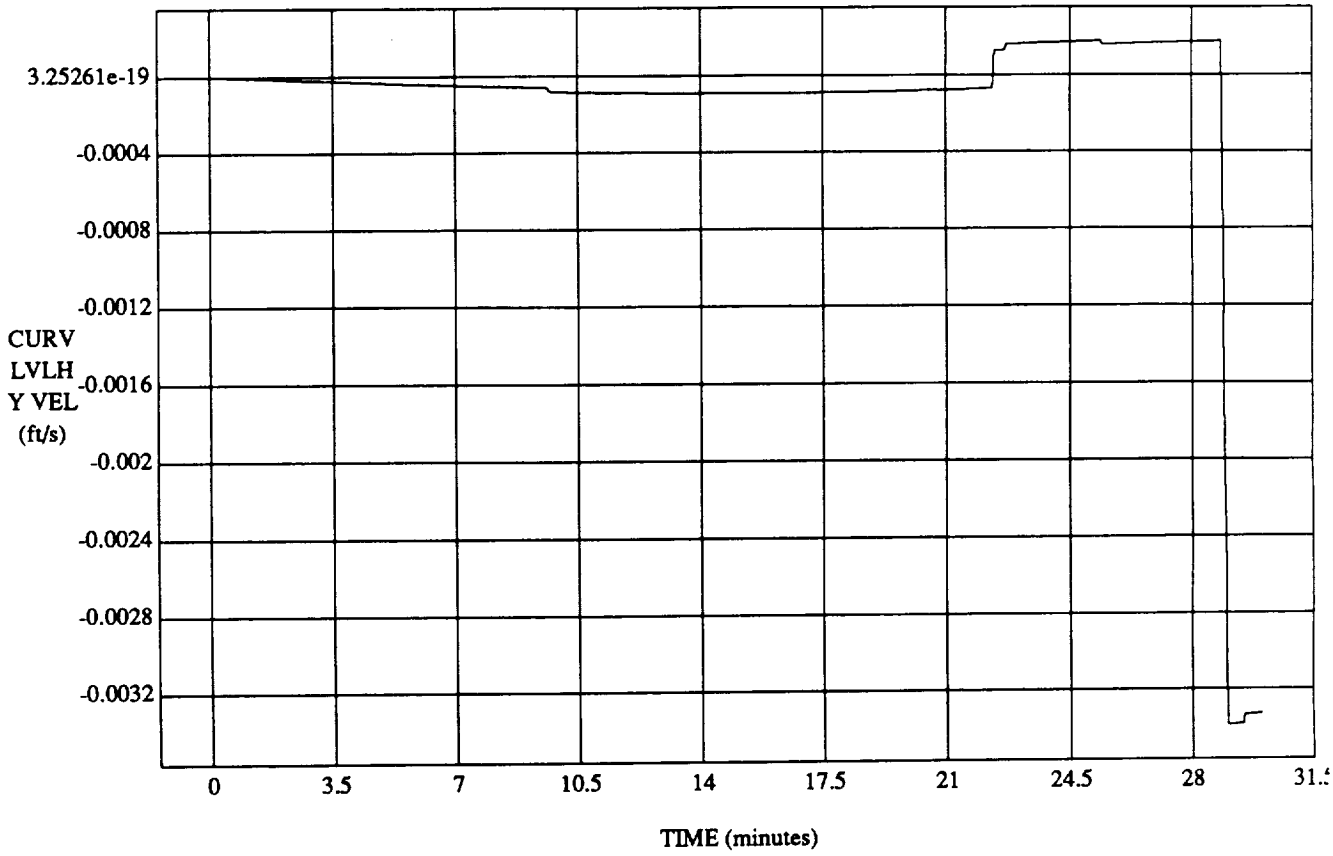
RUN: Station Keep At 200 Feet



VEHICLE: ORBITER.state
TARGET VEHICLE: SOLMAX.state
DATA SAMPLING FREQUENCY: 0.521 Hz

TARGET CENTERED ROTATING CURVILINEAR LVLH Y VEL vs TIME

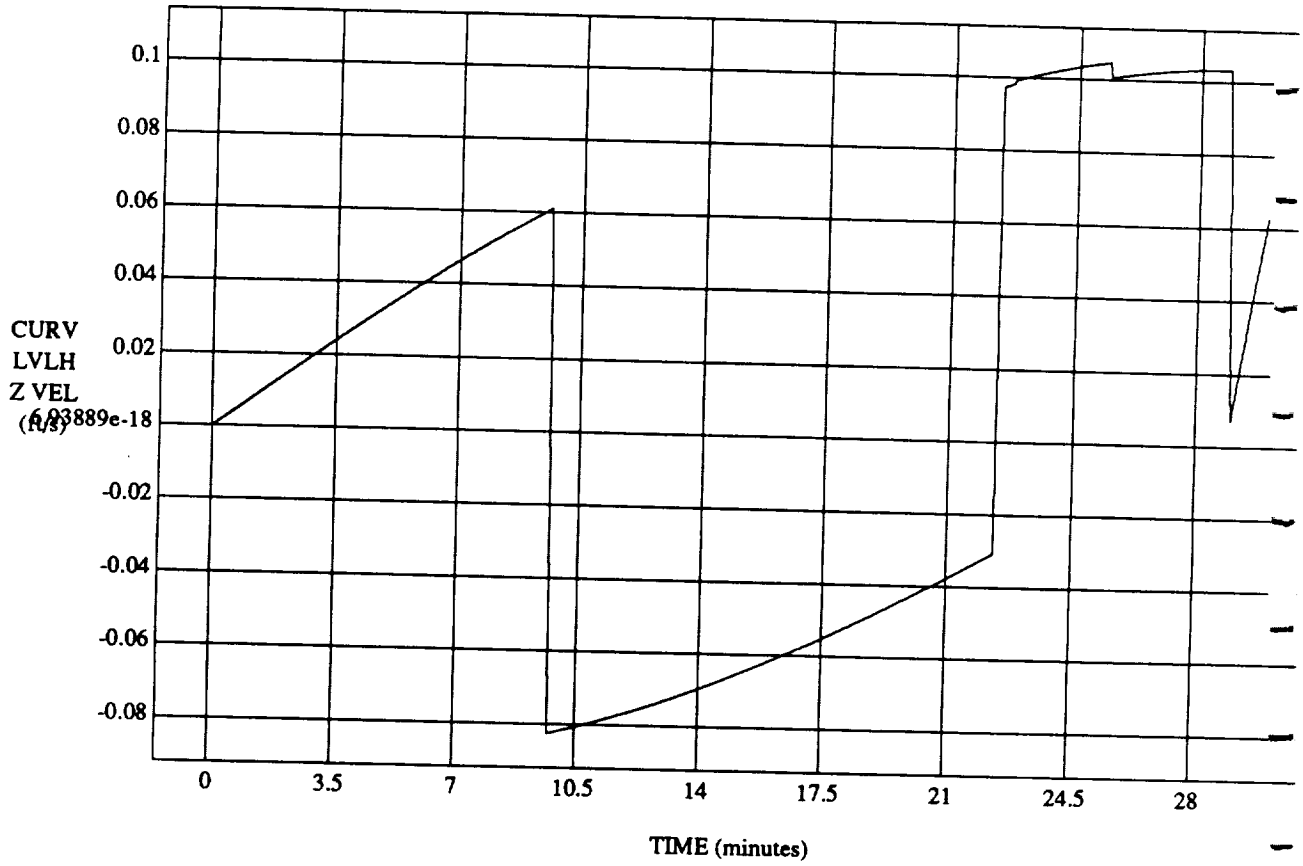
RUN: Station Keep At 200 Feet



VEHICLE: ORBITER.state
TARGET VEHICLE: SOLMAX.state
DATA SAMPLING FREQUENCY: 0.521 Hz

TARGET CENTERED ROTATING CURVILINEAR LVLH Z VEL vs TIME

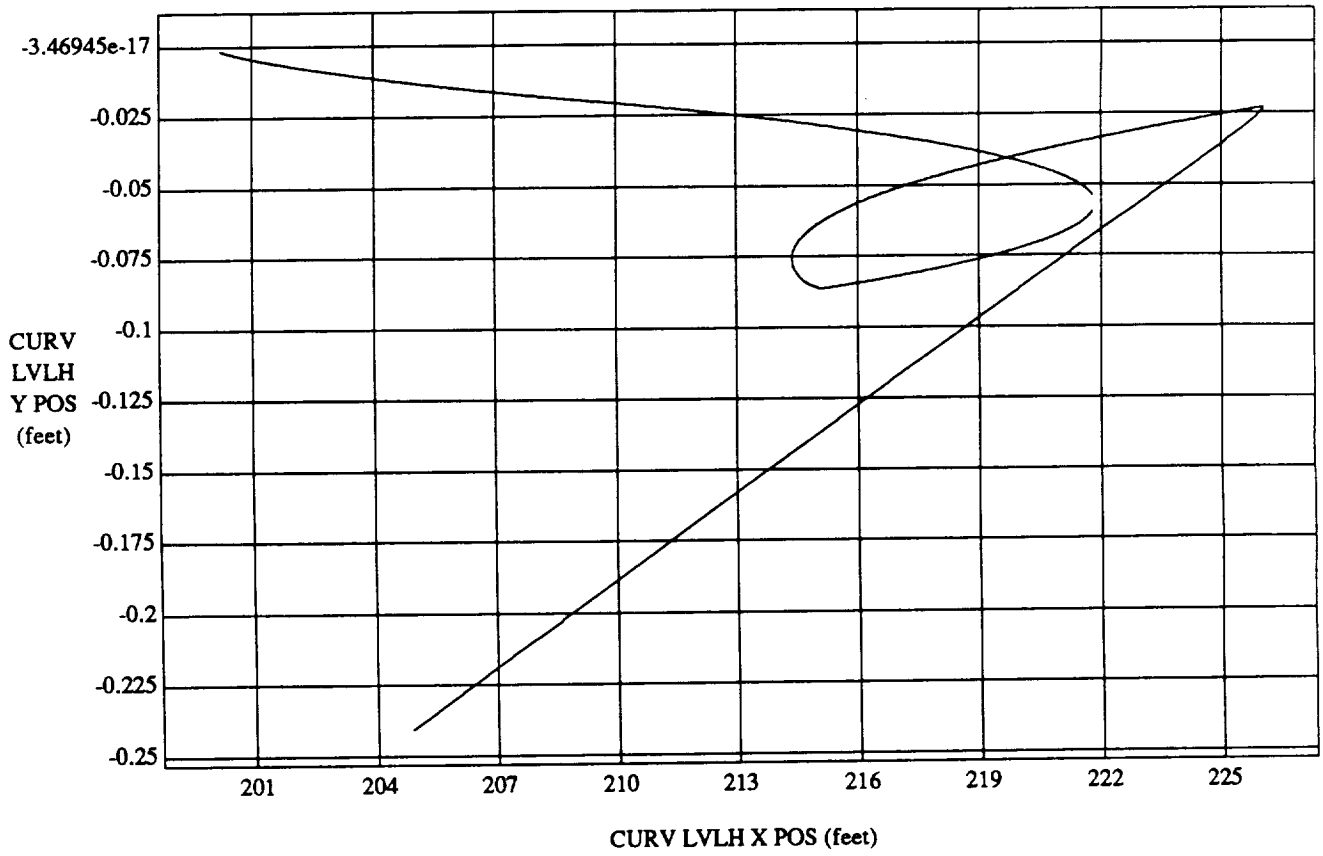
RUN: Station Keep At 200 Feet



VEHICLE: ORBITER.state
TARGET VEHICLE: SOLMAX.state
DATA SAMPLING FREQUENCY: 0.521 Hz

TARGET CENTERED ROTATING CURVILINEAR LVLH Y POS vs X POS

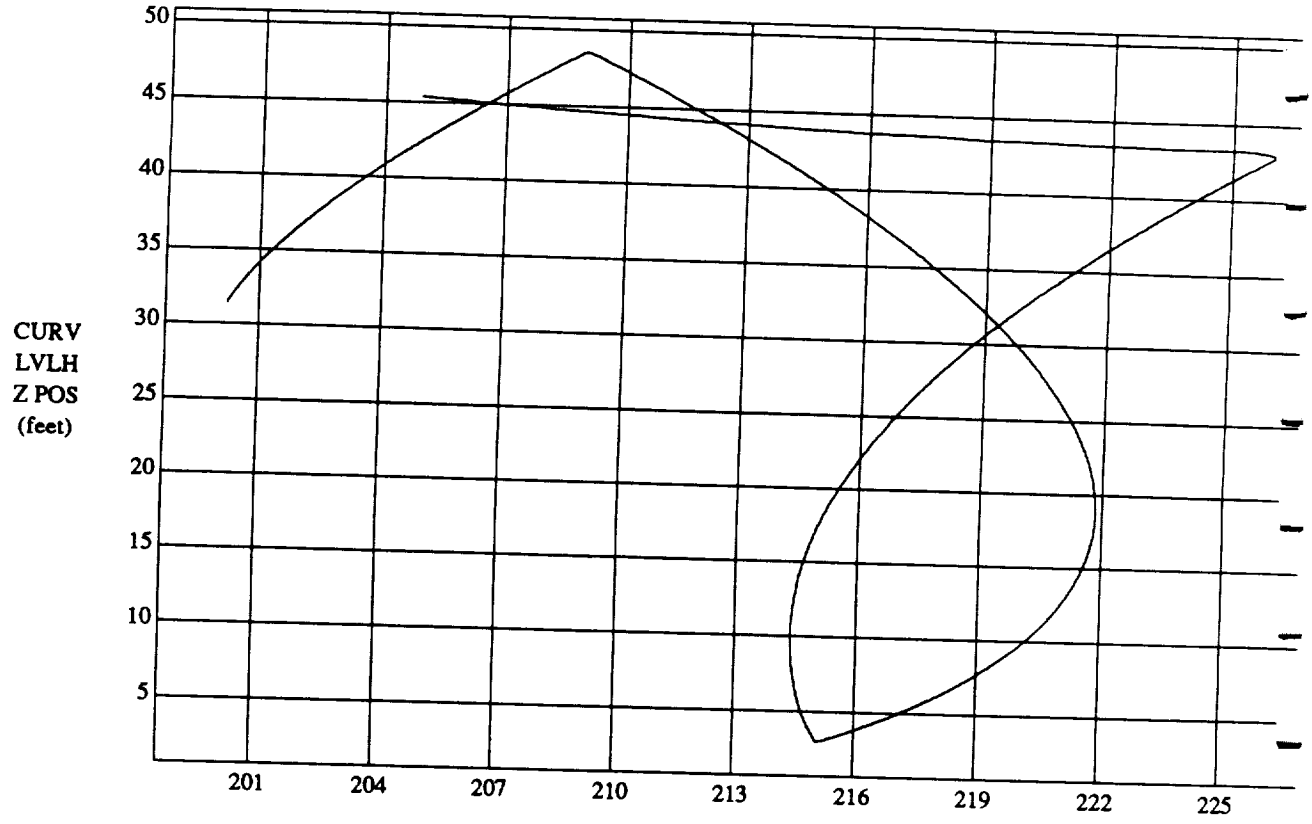
RUN: Station Keep At 200 Feet



VEHICLE: ORBITER.state
TARGET VEHICLE: SOLMAX.state
DATA SAMPLING FREQUENCY: 0.521 Hz

TARGET CENTERED ROTATING CURVILINEAR LVLH Z POS vs X POS

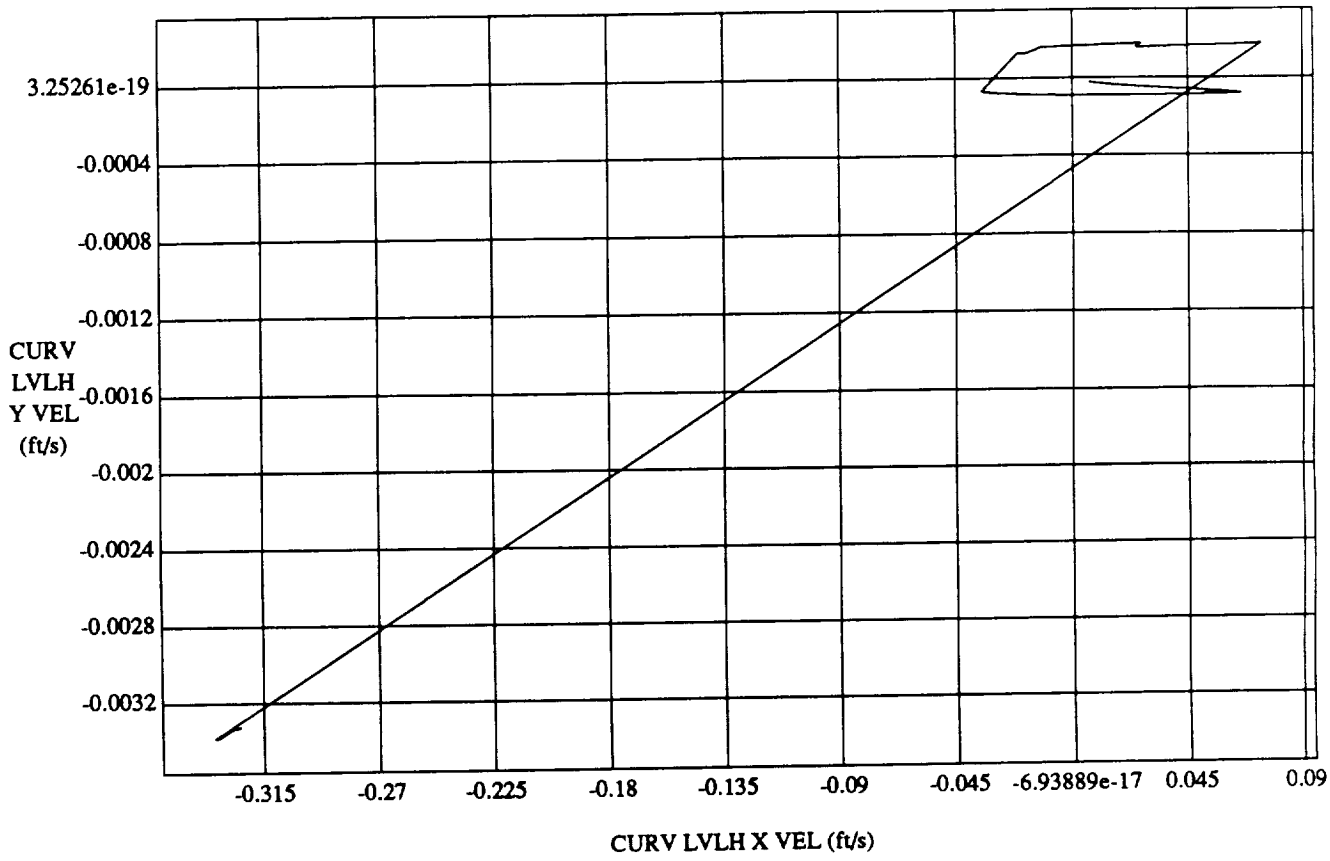
RUN: Station Keep At 200 Feet



VEHICLE: ORBITER.state
TARGET VEHICLE: SOLMAX.state
DATA SAMPLING FREQUENCY: 0.521 Hz

TARGET CENTERED ROTATING CURVILINEAR LVLH Y VEL vs X VEL

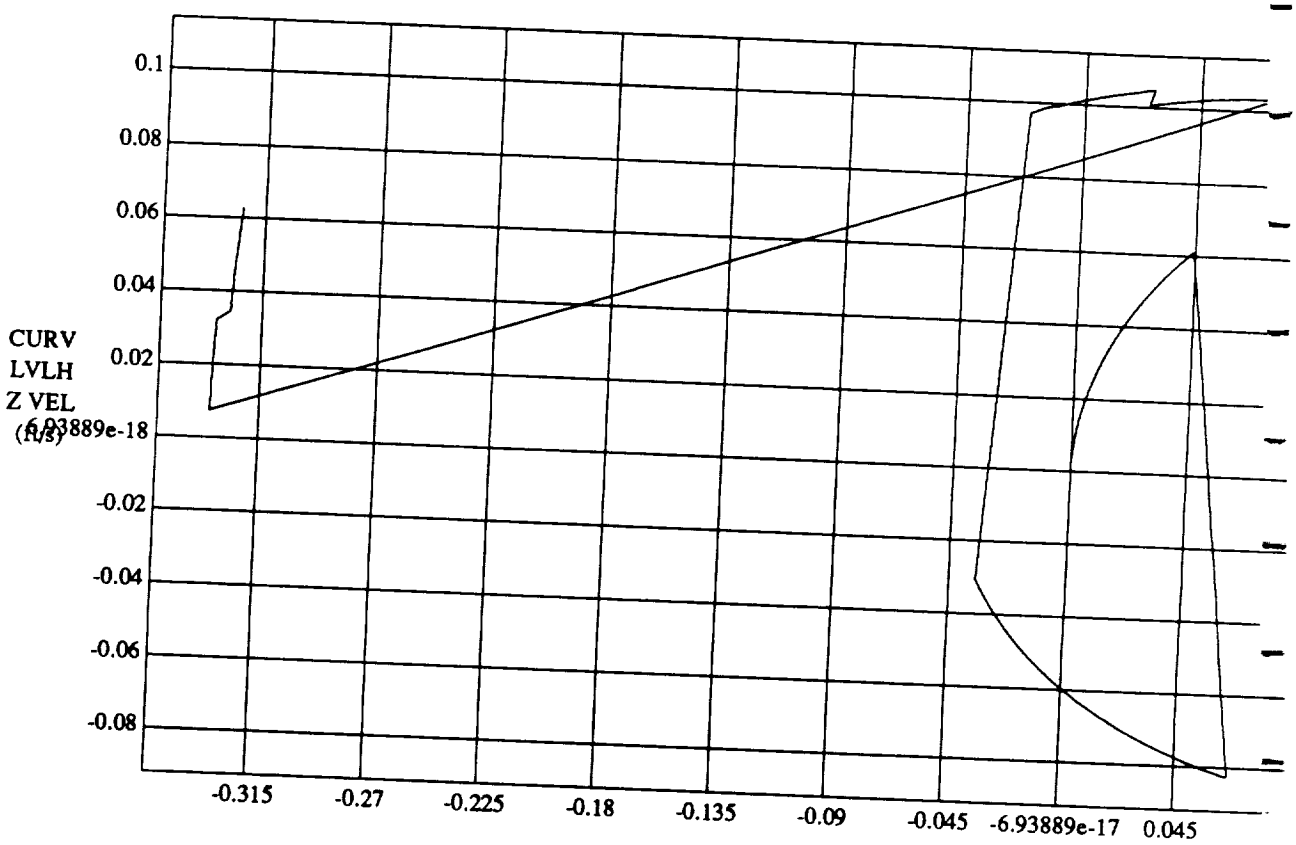
RUN: Station Keep At 200 Feet



VEHICLE: ORBITER.state
TARGET VEHICLE: SOLMAX.state
DATA SAMPLING FREQUENCY: 0.521 Hz

TARGET CENTERED ROTATING CURVILINEAR LVLH Z VEL vs X VEL

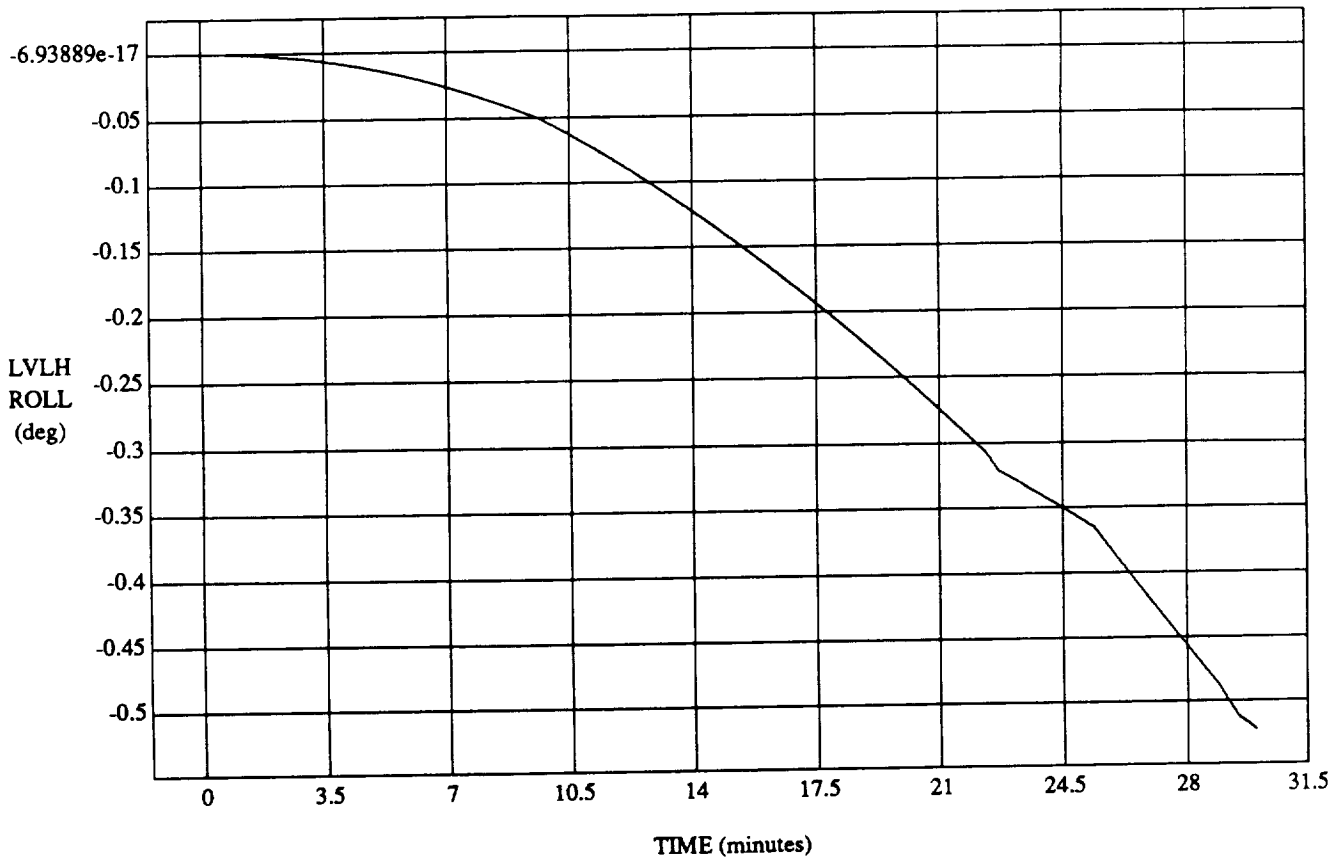
RUN: Station Keep At 200 Feet



VEHICLE: ORBITER.state
TARGET VEHICLE: SOLMAX.state
DATA SAMPLING FREQUENCY: 0.521 Hz

LVLH EULER PYR ROLL vs TIME

RUN: Station Keep At 200 Feet

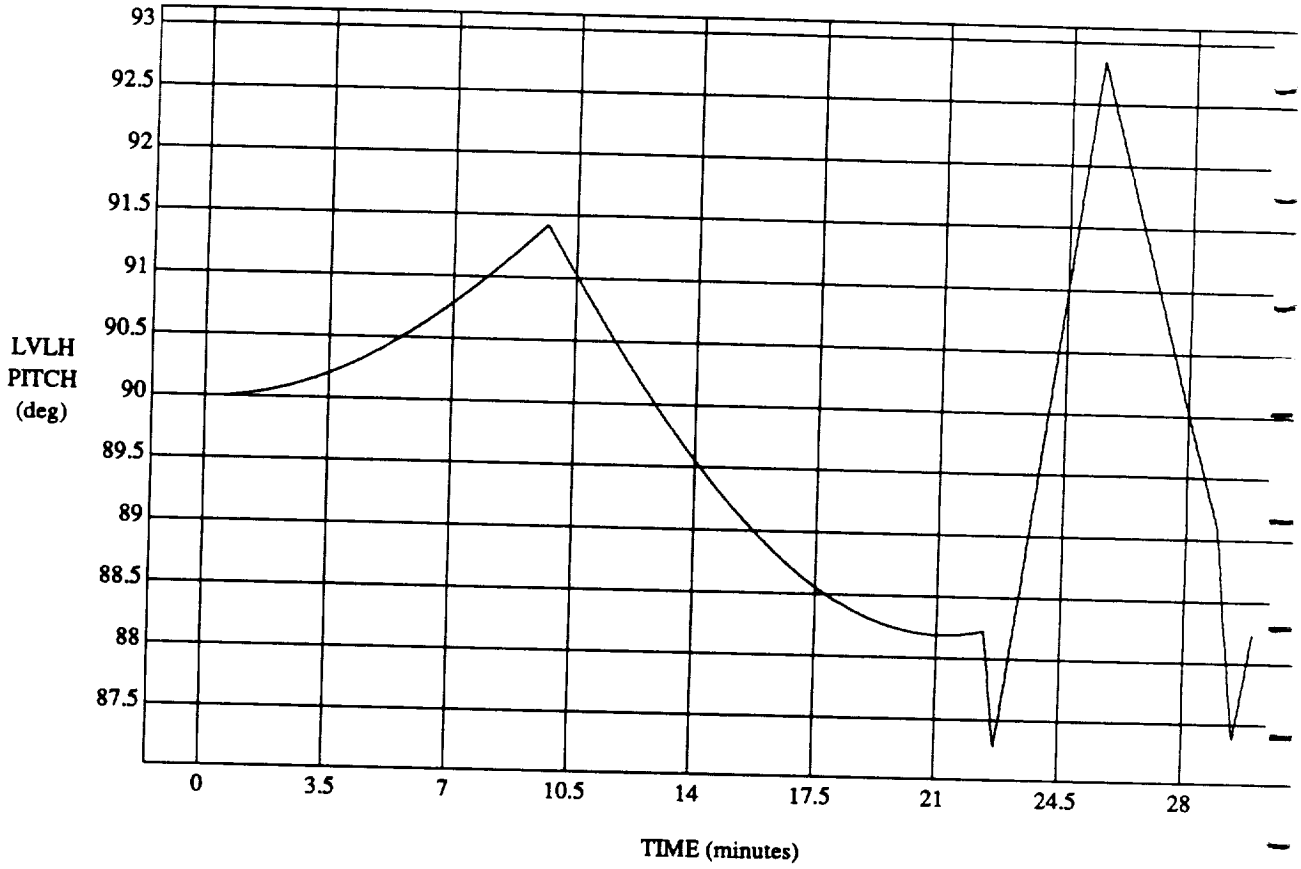


VEHICLE: ORBITER.state

DATA SAMPLING FREQUENCY: 0.521 Hz

LVLH EULER PYR PITCH vs TIME

RUN: Station Keep At 200 Feet

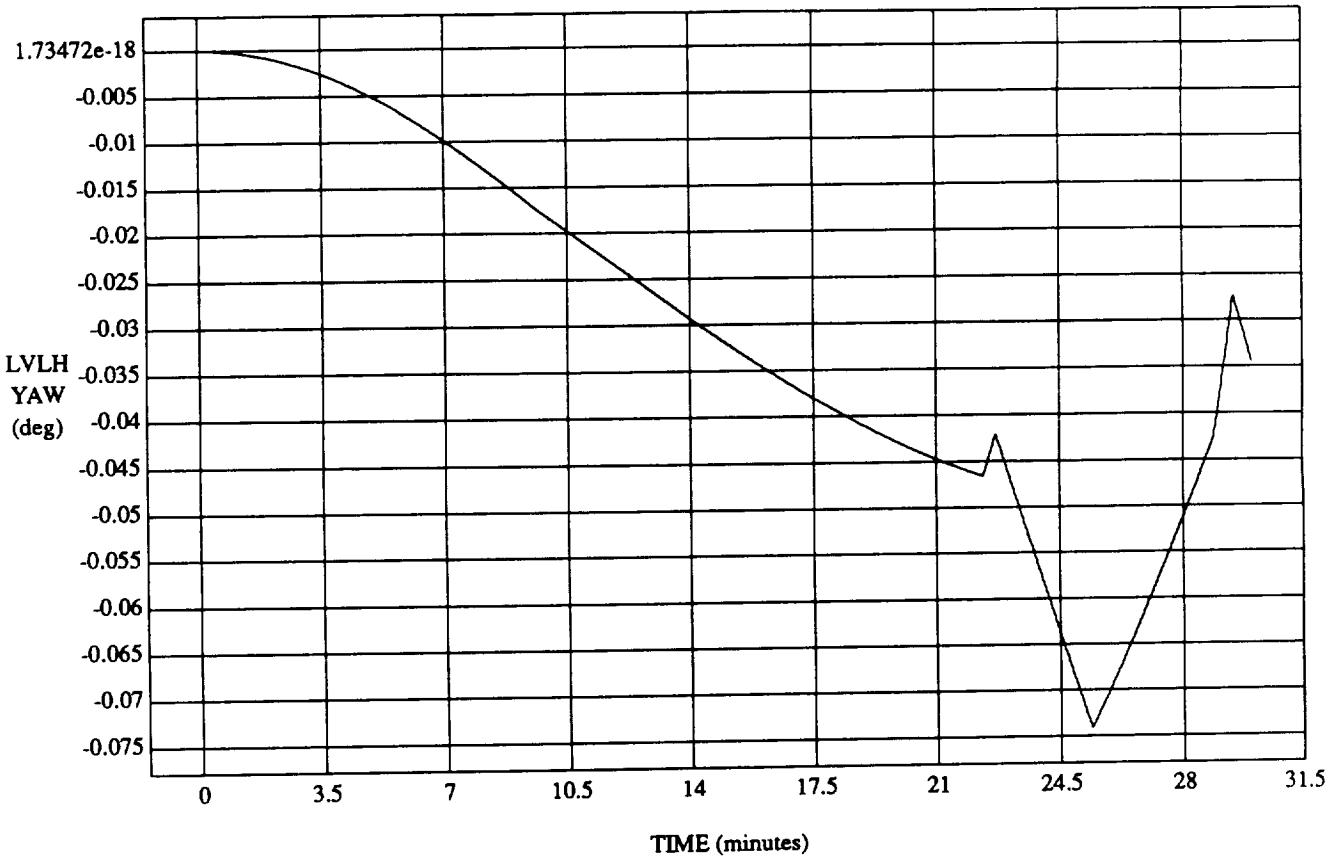


VEHICLE: ORBITER.state

DATA SAMPLING FREQUENCY: 0.521 Hz

LVLH EULER PYR YAW vs TIME

RUN: Station Keep At 200 Feet

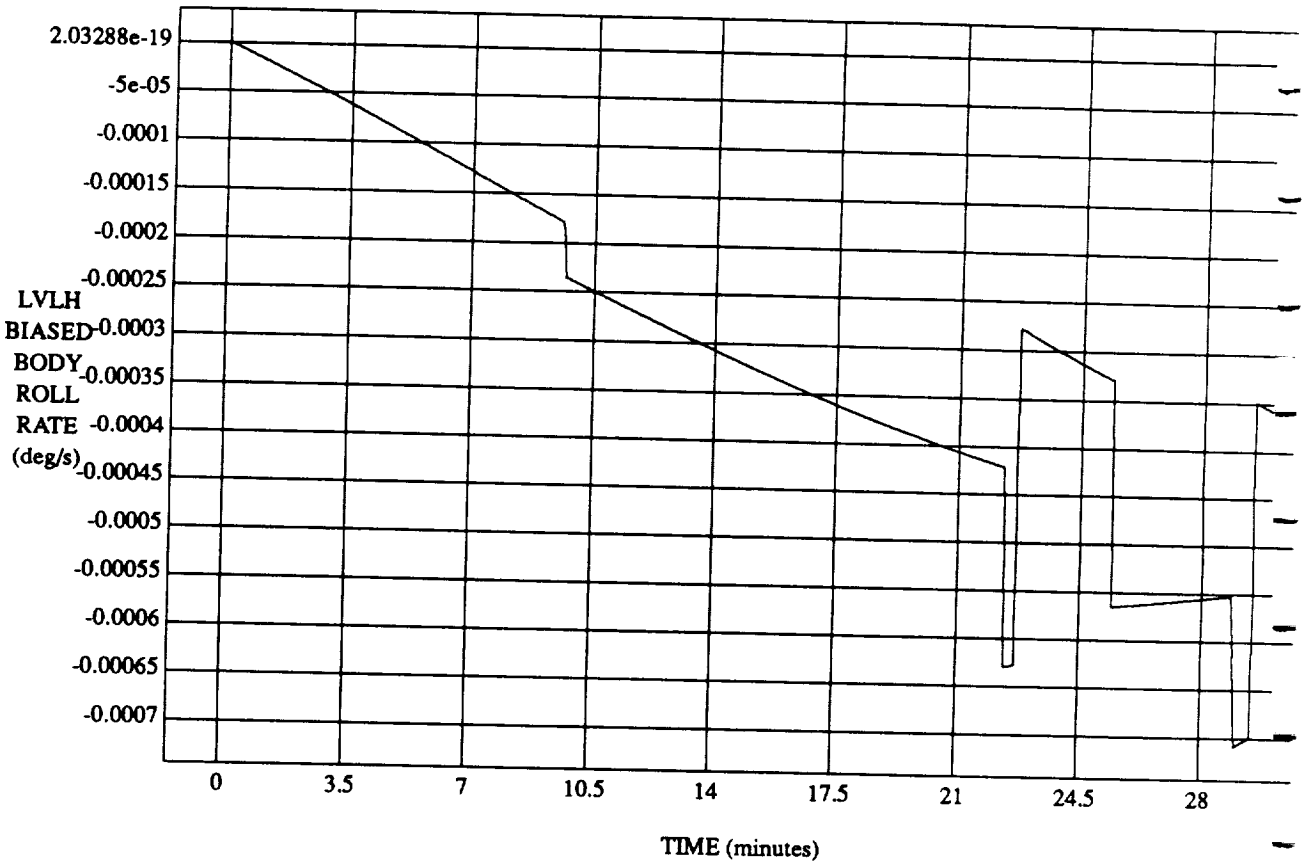


VEHICLE: ORBITER.state

DATA SAMPLING FREQUENCY: 0.521 Hz

LVLH BIASED BODY ROLL RATE vs TIME

RUN: Station Keep At 200 Feet

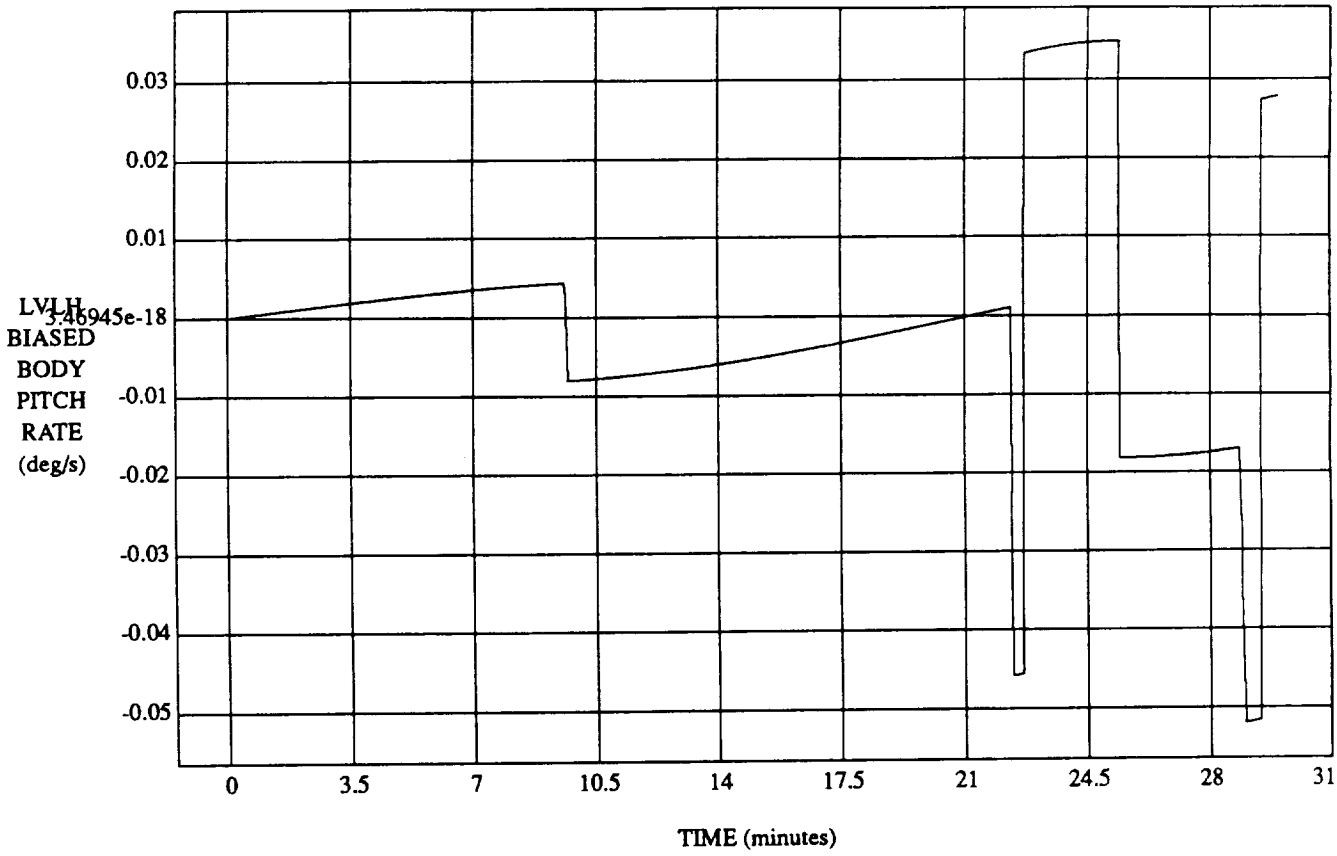


VEHICLE: ORBITER.state

DATA SAMPLING FREQUENCY: 0.521 Hz

LVLH BIASED BODY PITCH RATE vs TIME

RUN: Station Keep At 200 Feet

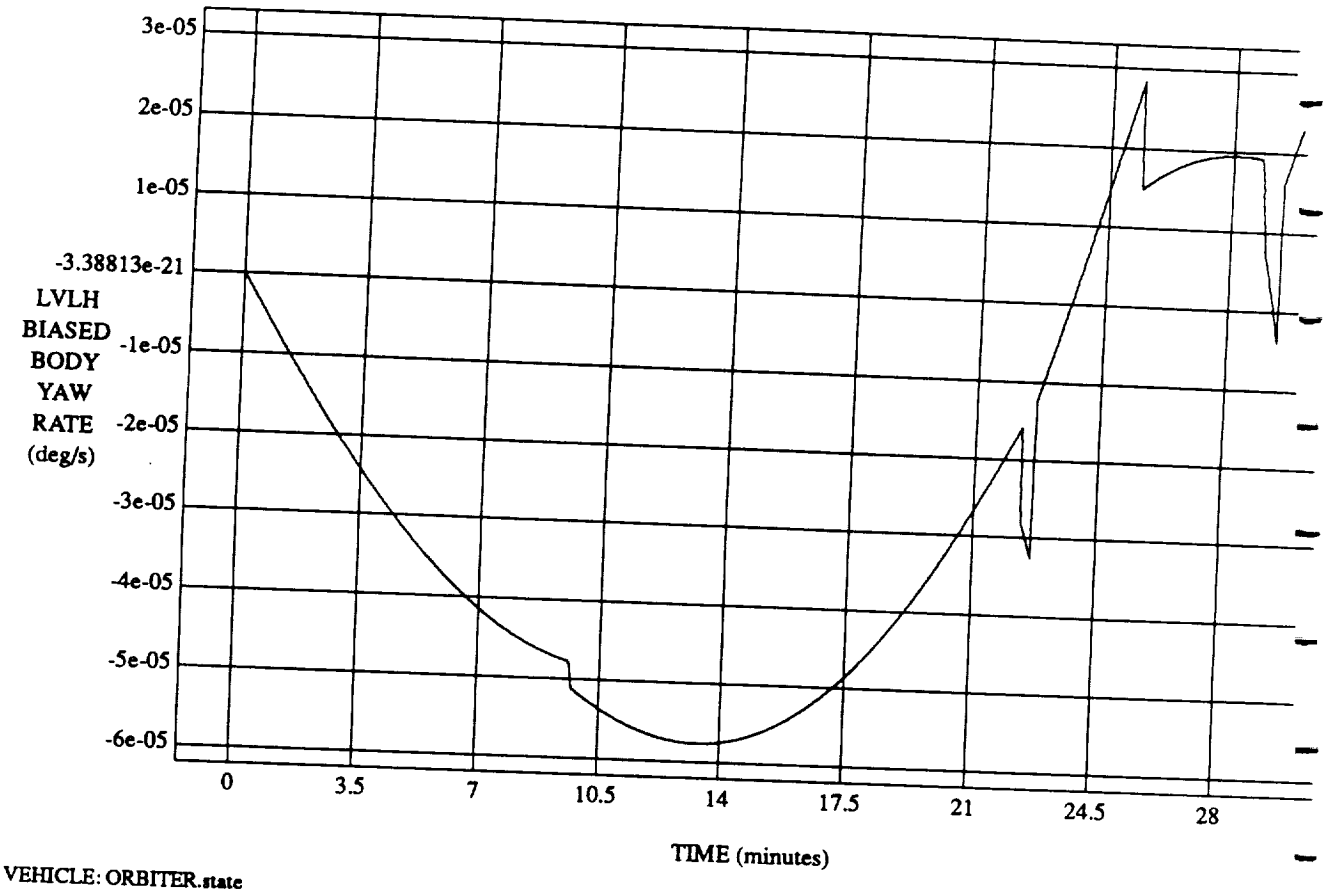


VEHICLE: ORBITER.state

DATA SAMPLING FREQUENCY: 0.521 Hz

LVLH BIASED BODY YAW RATE vs TIME

RUN: Station Keep At 200 Feet



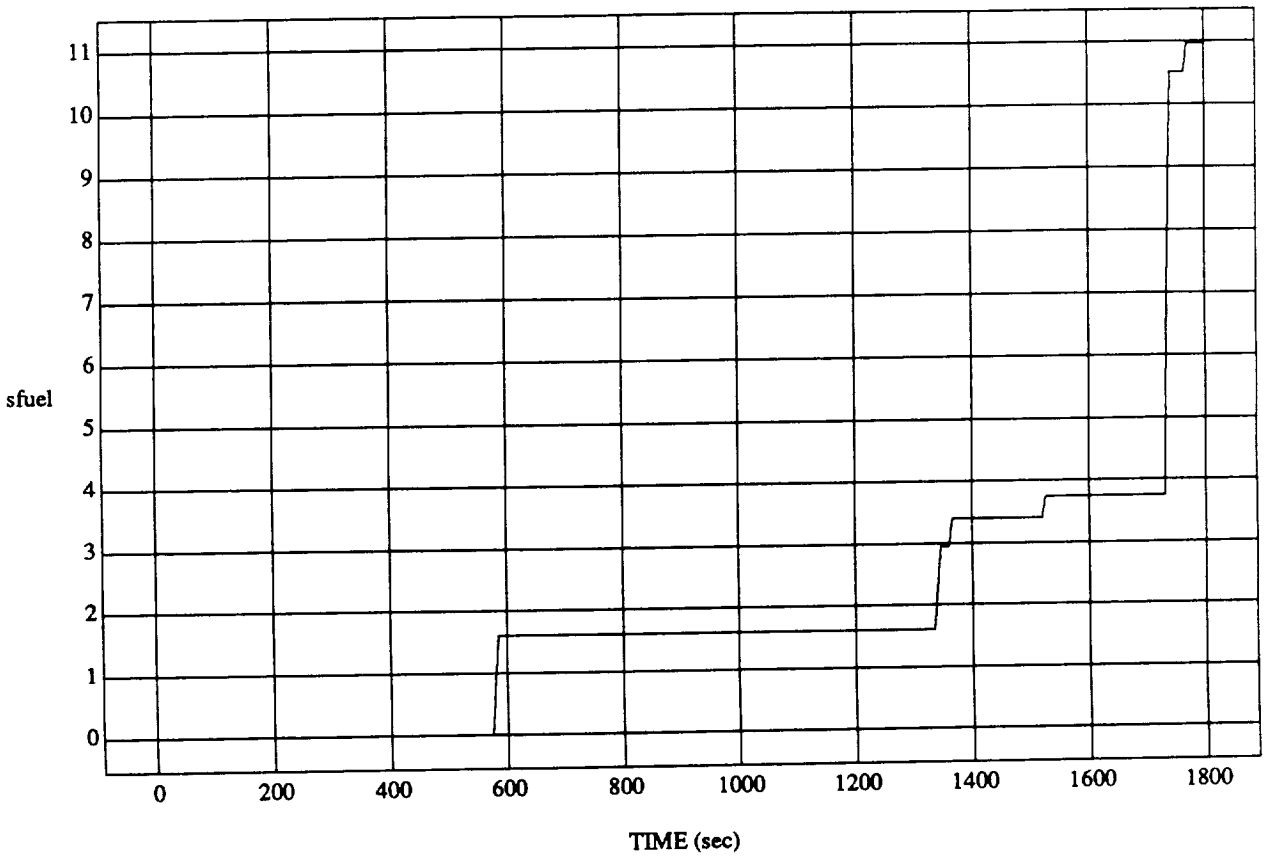
VEHICLE: ORBITER.state

DATA SAMPLING FREQUENCY: 0.521 Hz

SIMULATION APPLICATION: ARIC Translational Controller Simulation

sfuel vs TIME

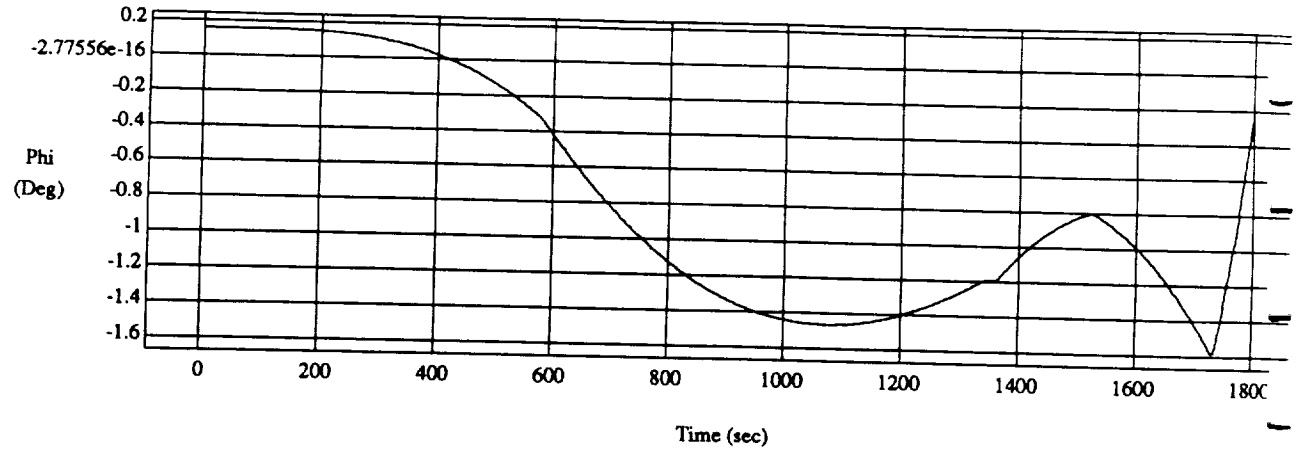
RUN: Station Keep At 200 Feet

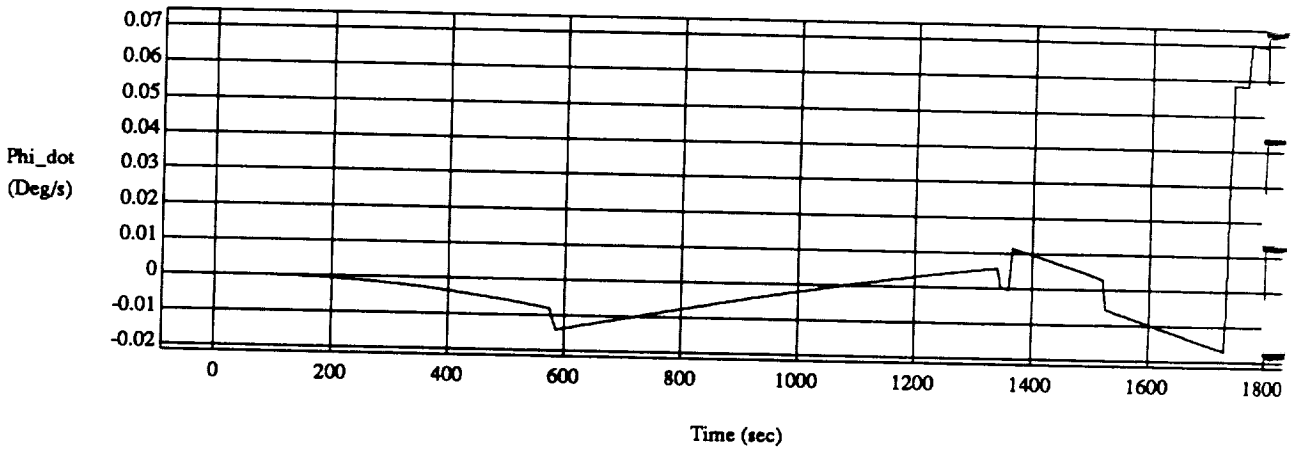


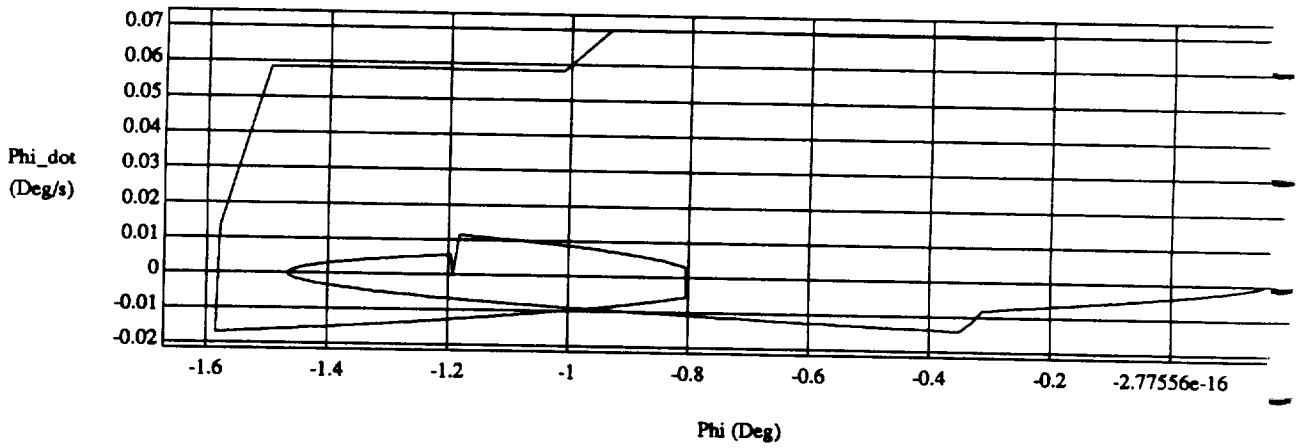
MODULE: ORBITER.primary

DATA SAMPLING FREQUENCY: 0.200 Hz

Range ARIC Learning Parameters - Inputs

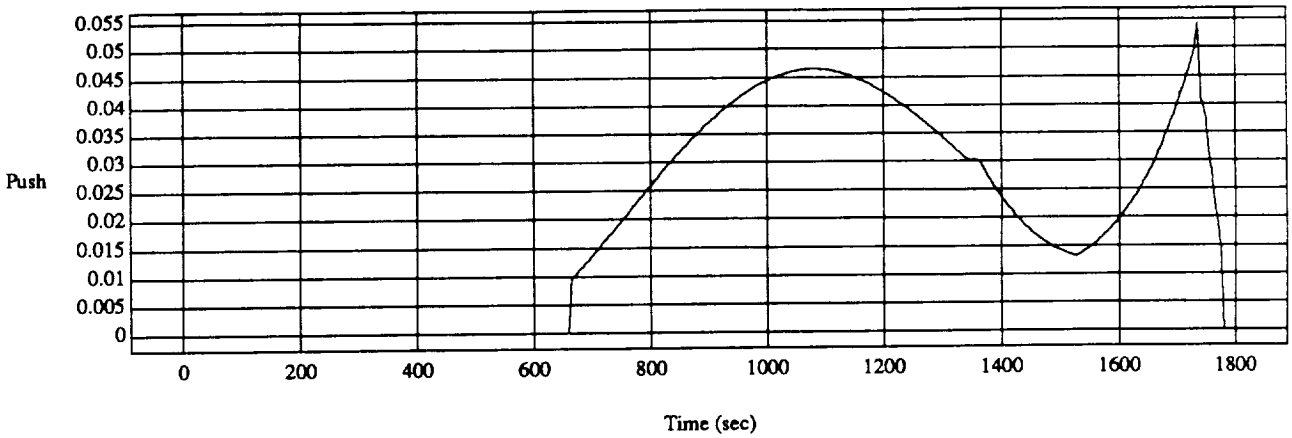


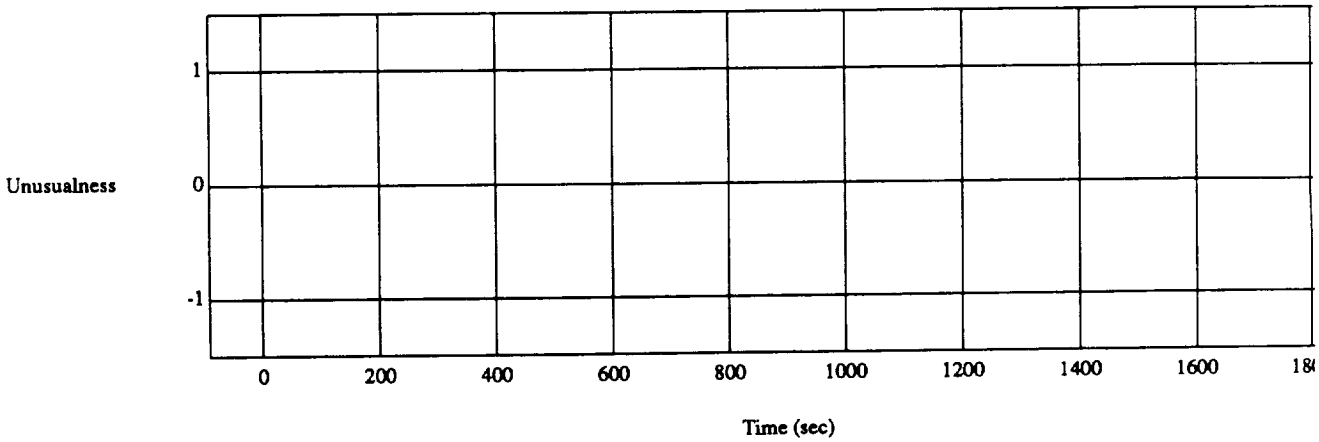


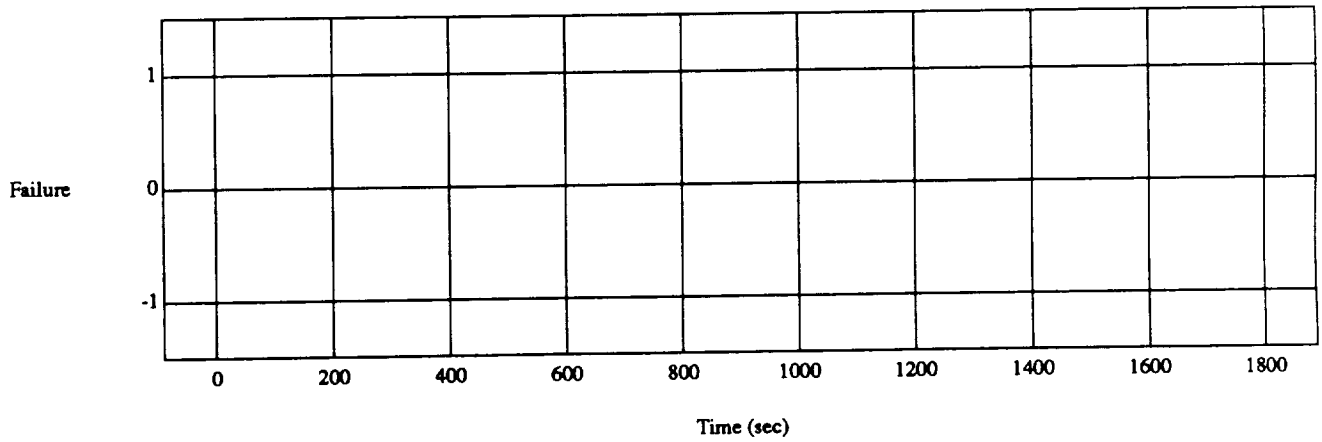


SIMULATION APPLICATION: ARIC Translational Controller Simulation
RUN IDENTIFICATION: Station Keep At 200 Feet
MODEL: ORBITER
DATE: Sat Nov 21 1992 04:20:52 PM
NUMBER OF DATA POINTS: 361
DATA SAMPLING FREQUENCY: 0.200 Hz

Range ARIC Learning Parameters - General Parameters

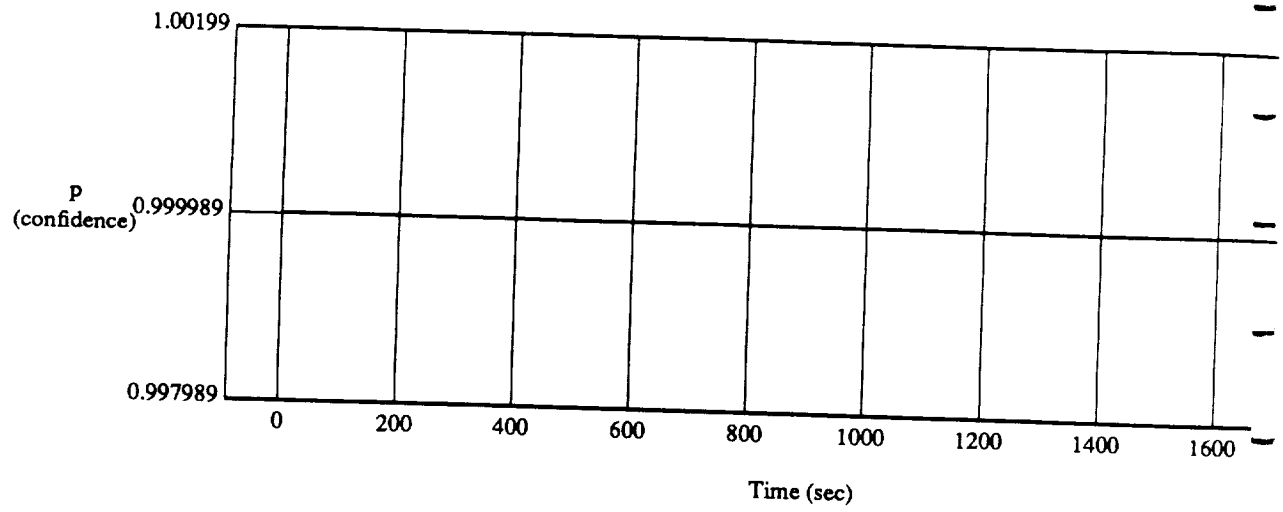


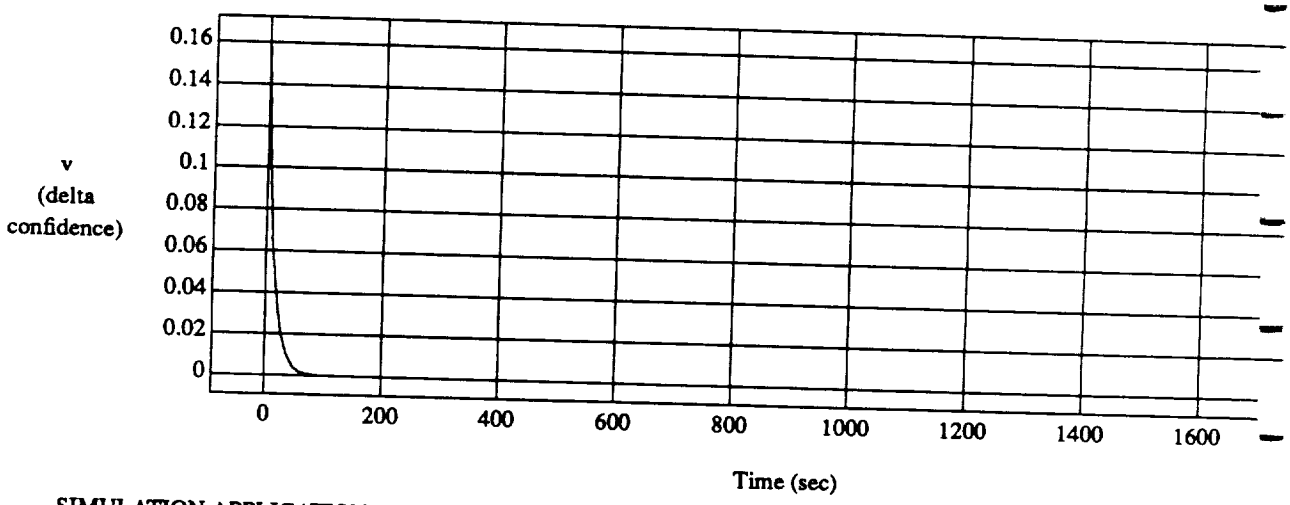




SIMULATION APPLICATION: ARIC Translational Controller Simulation
RUN IDENTIFICATION: Station Keep At 200 Feet
MODEL: ORBITER
DATE: Sat Nov 21 1992 04:20:52 PM
NUMBER OF DATA POINTS: 361
DATA SAMPLING FREQUENCY: 0.200 Hz

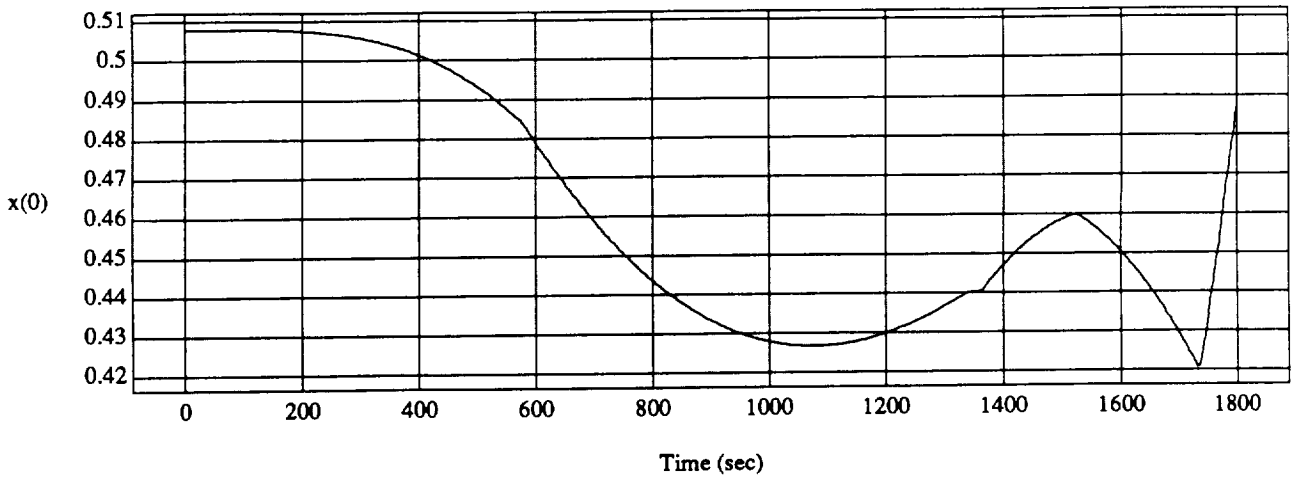
Range ARIC Learning Parameters - Learning Confidence

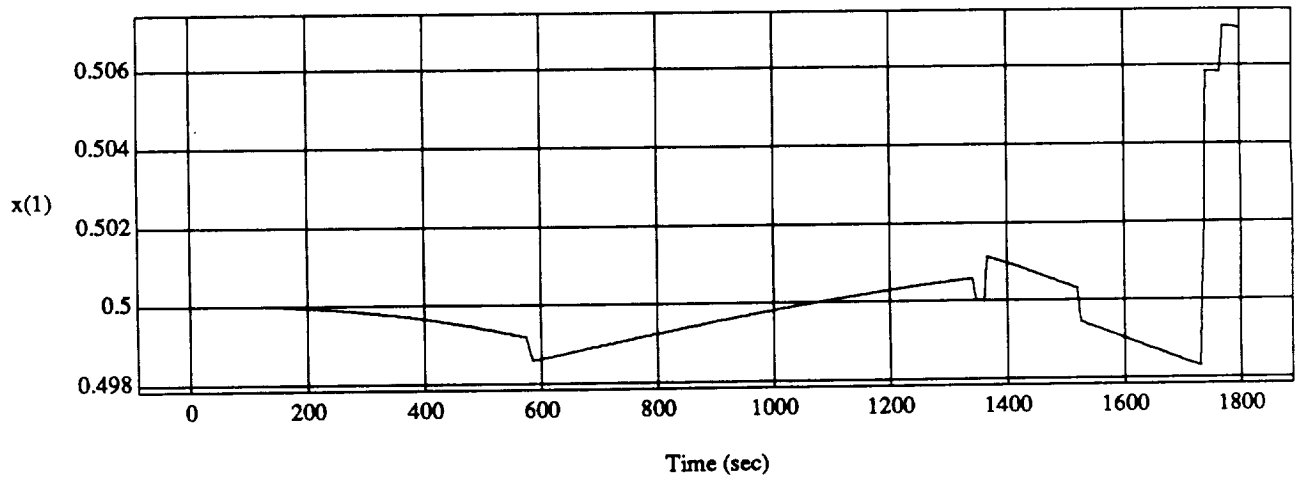




SIMULATION APPLICATION: ARIC Translational Controller Simulation
RUN IDENTIFICATION: Station Keep At 200 Feet
MODEL: ORBITER
DATE: Sat Nov 21 1992 04:20:52 PM
NUMBER OF DATA POINTS: 361
DATA SAMPLING FREQUENCY: 0.200 Hz

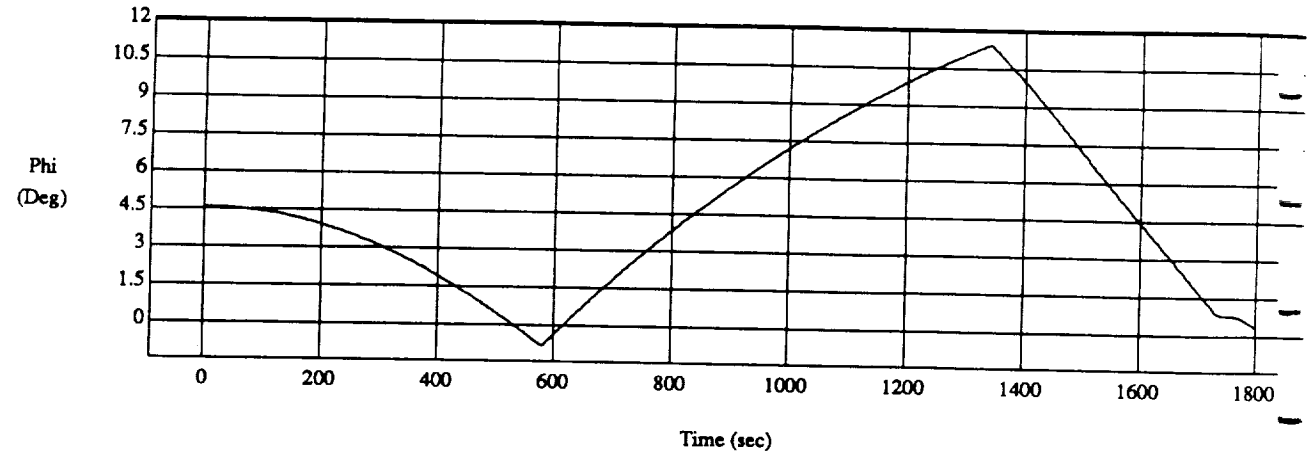
Range ARIC Learning Parameters - Scaled Inputs

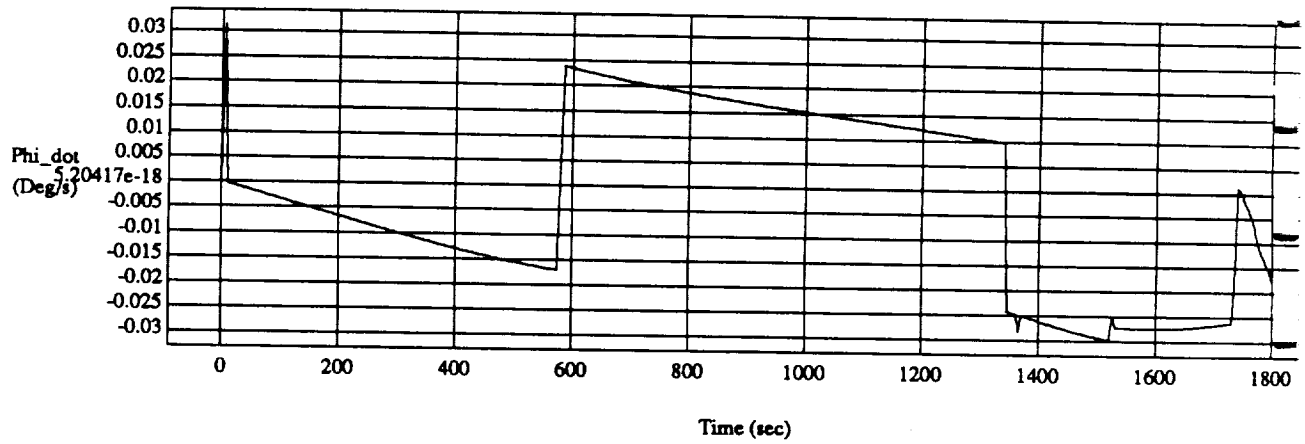


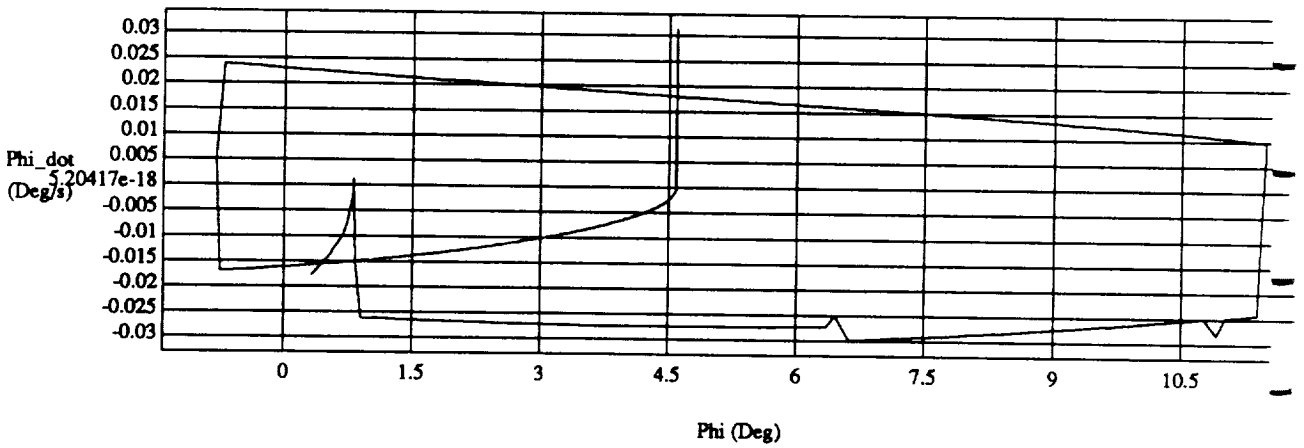


SIMULATION APPLICATION: ARIC Translational Controller Simulation
RUN IDENTIFICATION: Station Keep At 200 Feet
MODEL: ORBITER
DATE: Sat Nov 21 1992 04:20:52 PM
NUMBER OF DATA POINTS: 361
DATA SAMPLING FREQUENCY: 0.200 Hz

Elevation ARIC Learning Parameters - Inputs

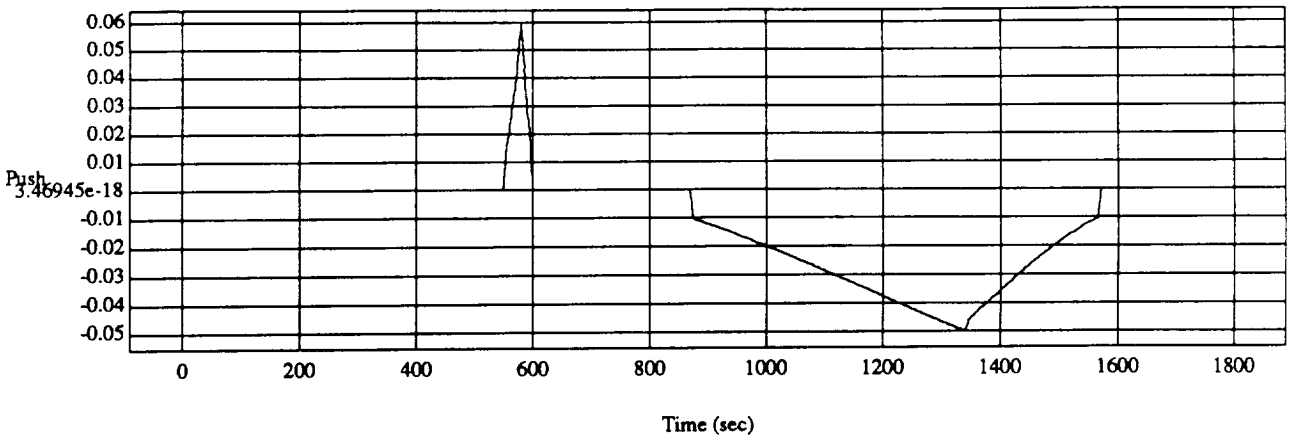


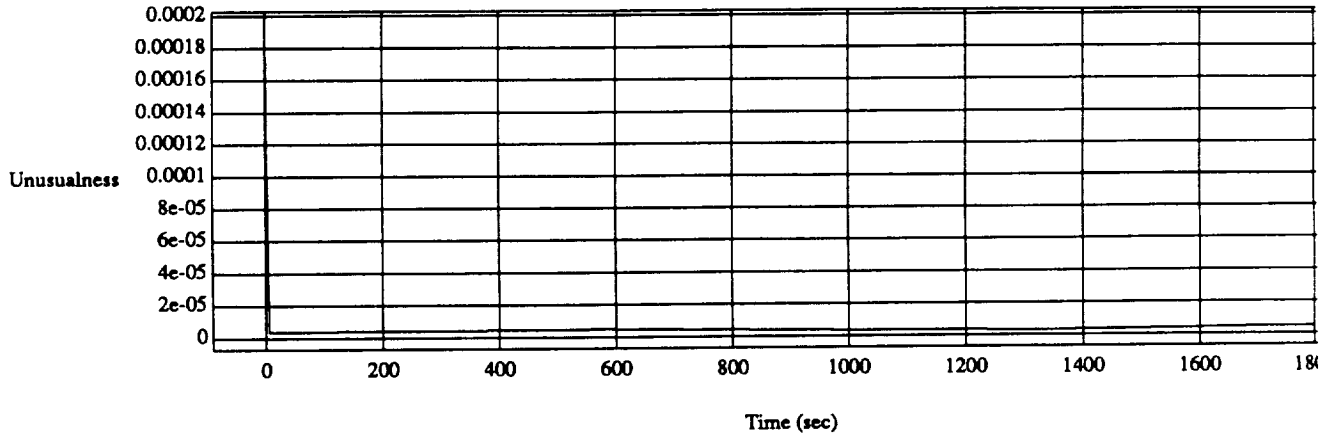


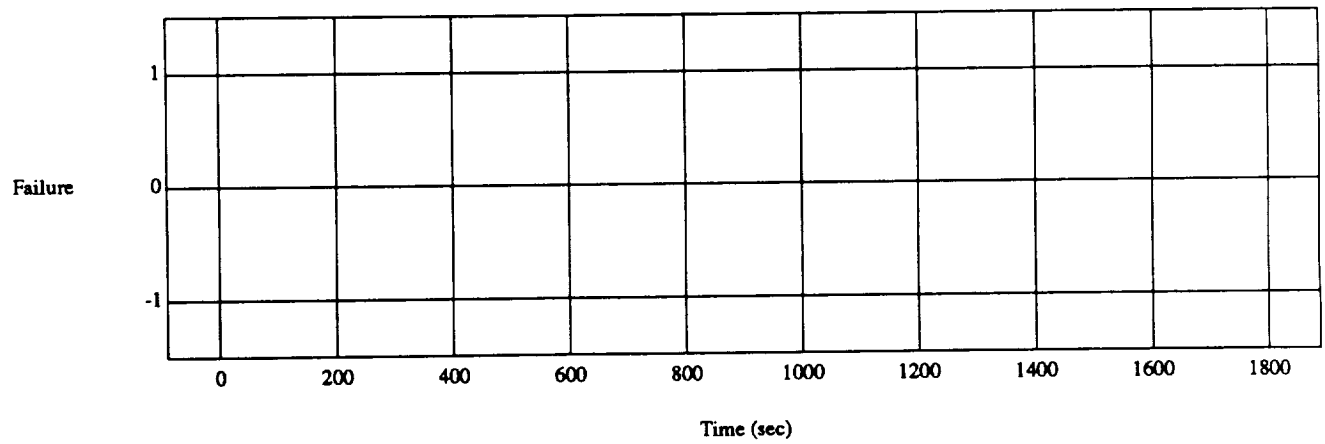


SIMULATION APPLICATION: ARIC Translational Controller Simulation
RUN IDENTIFICATION: Station Keep At 200 Feet
MODEL: ORBITER
DATE: Sat Nov 21 1992 04:20:35 PM
NUMBER OF DATA POINTS: 361
DATA SAMPLING FREQUENCY: 0.200 Hz

Elevation ARIC Learning Parameters - General Parameters

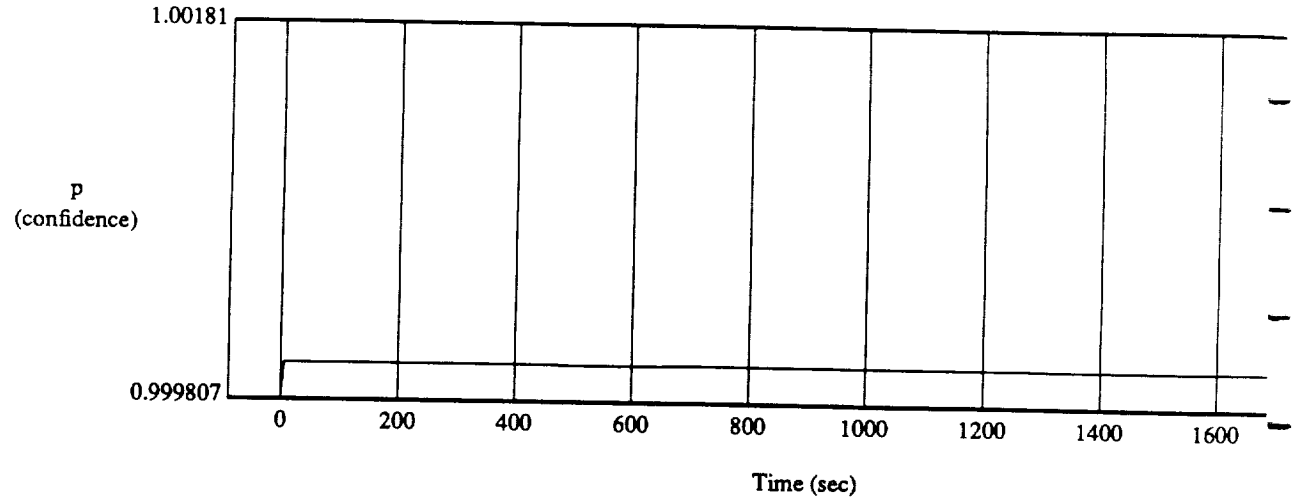


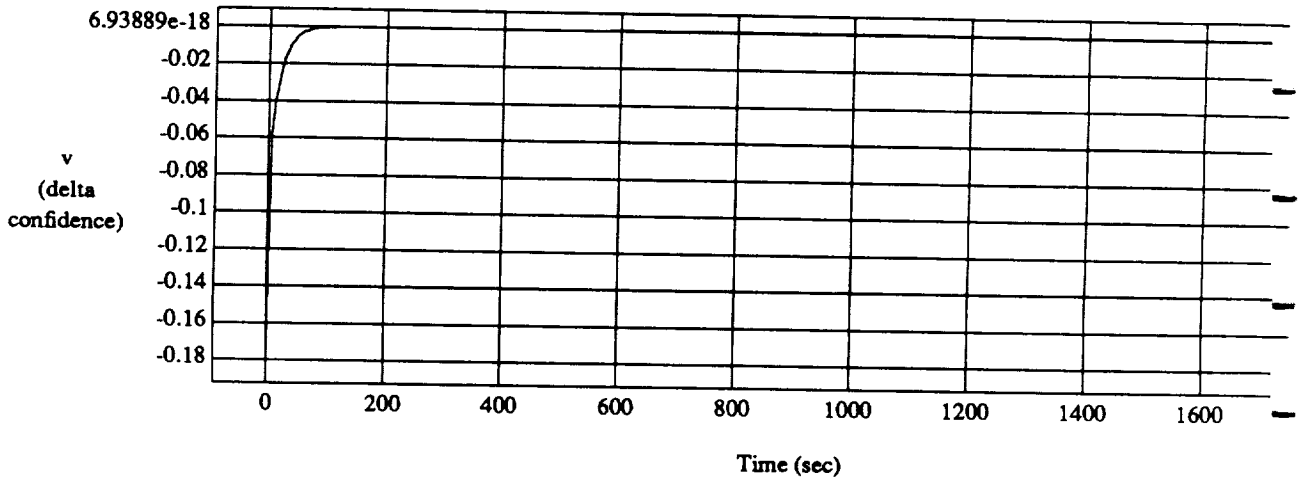




SIMULATION APPLICATION: ARIC Translational Controller Simulation
RUN IDENTIFICATION: Station Keep At 200 Feet
MODEL: ORBITER
DATE: Sat Nov 21 1992 04:20:35 PM
NUMBER OF DATA POINTS: 361
DATA SAMPLING FREQUENCY: 0.200 Hz

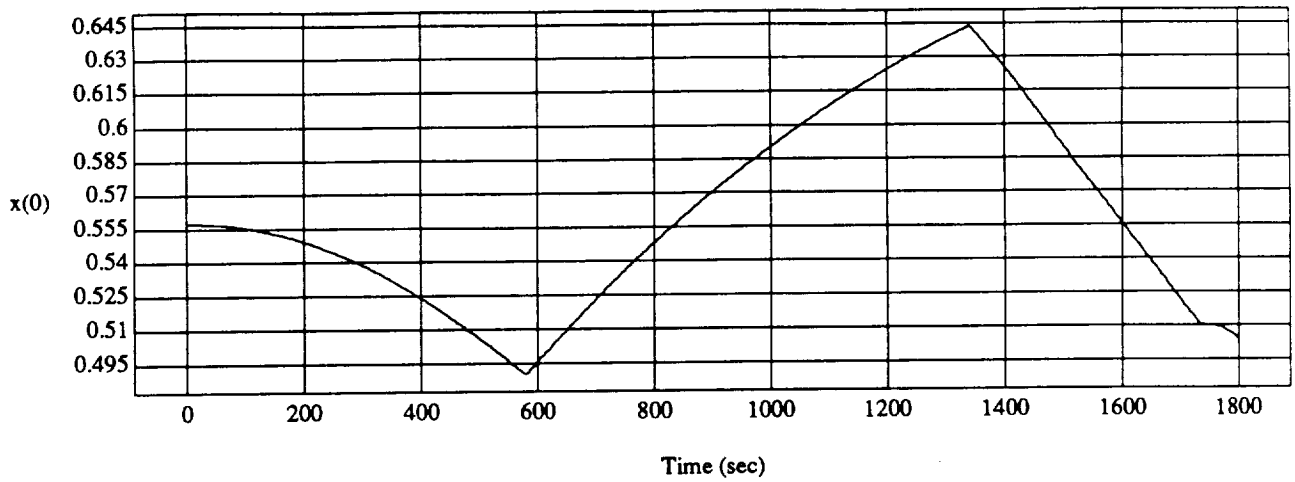
Elevation ARIC Learning Parameters - Learning Confidence

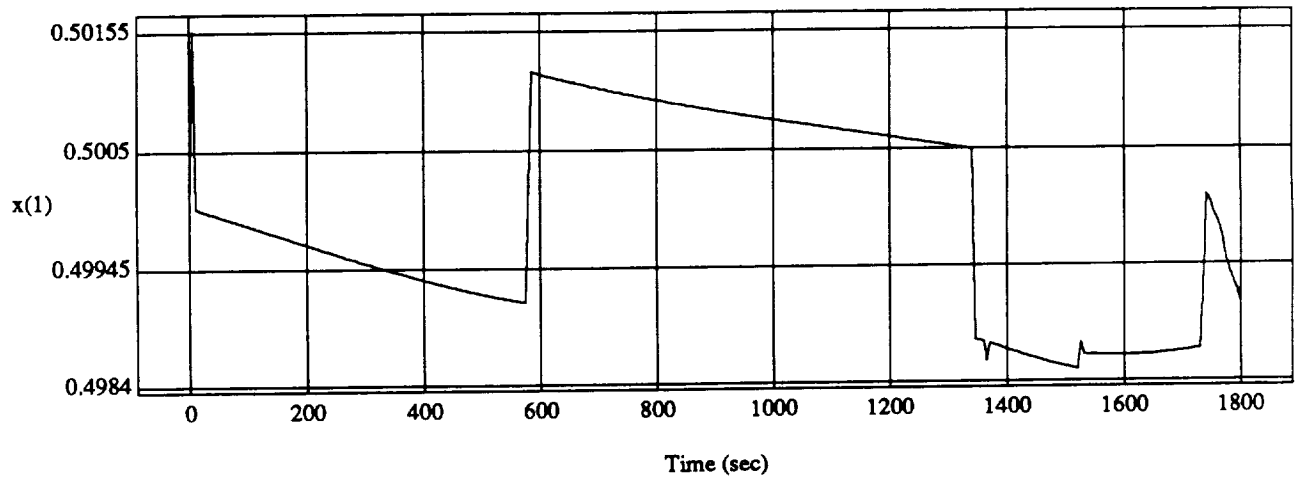




SIMULATION APPLICATION: ARIC Translational Controller Simulation
RUN IDENTIFICATION: Station Keep At 200 Feet
MODEL: ORBITER
DATE: Sat Nov 21 1992 04:20:35 PM
NUMBER OF DATA POINTS: 361
DATA SAMPLING FREQUENCY: 0.200 Hz

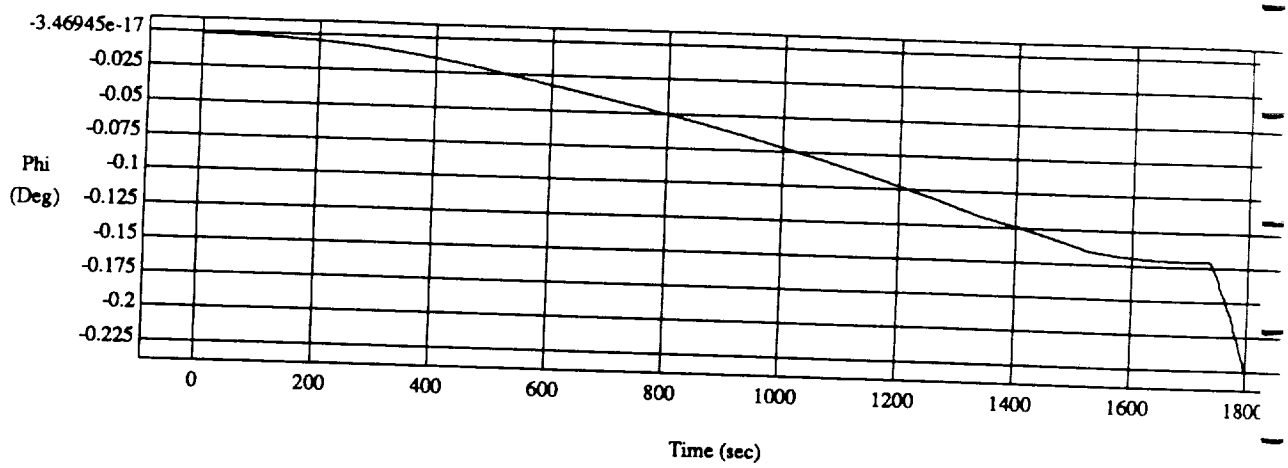
Elevation ARIC Learning Parameters - Scaled Inputs

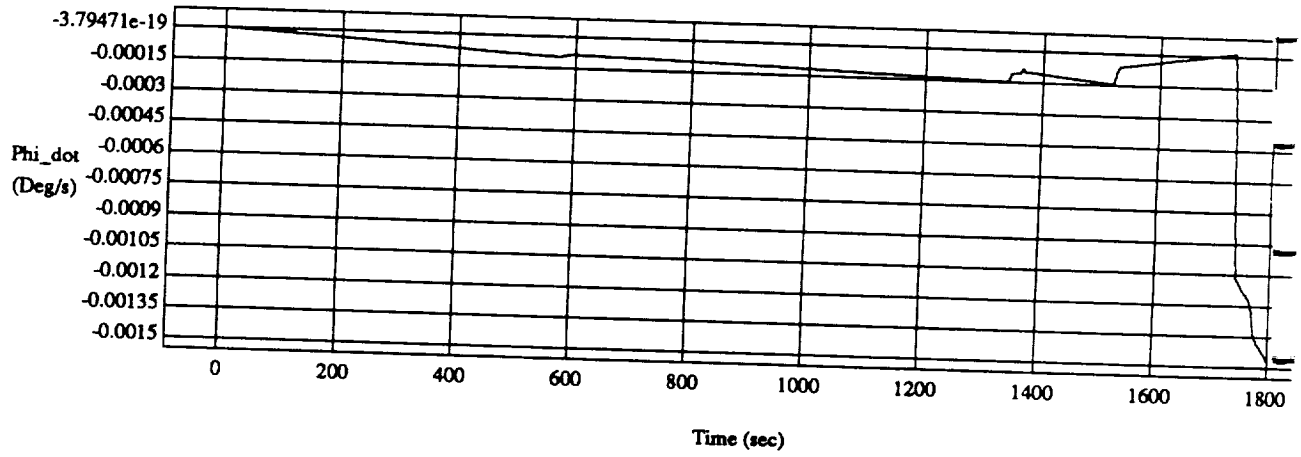


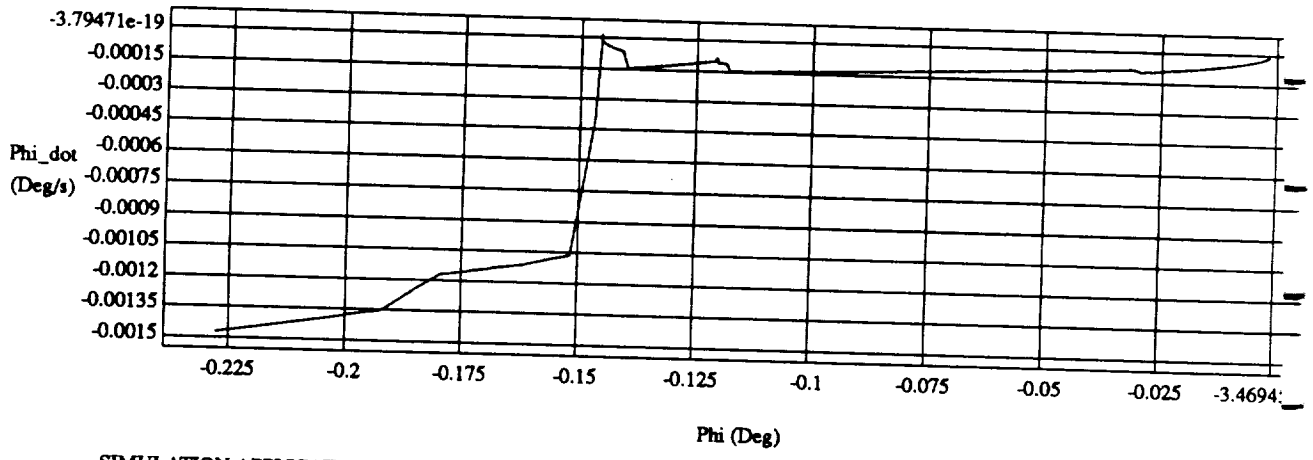


SIMULATION APPLICATION: ARIC Translational Controller Simulation
RUN IDENTIFICATION: Station Keep At 200 Feet
MODEL: ORBITER
DATE: Sat Nov 21 1992 04:20:35 PM
NUMBER OF DATA POINTS: 361
DATA SAMPLING FREQUENCY: 0.200 Hz

Azimuth ARIC Learning Parameters - Inputs

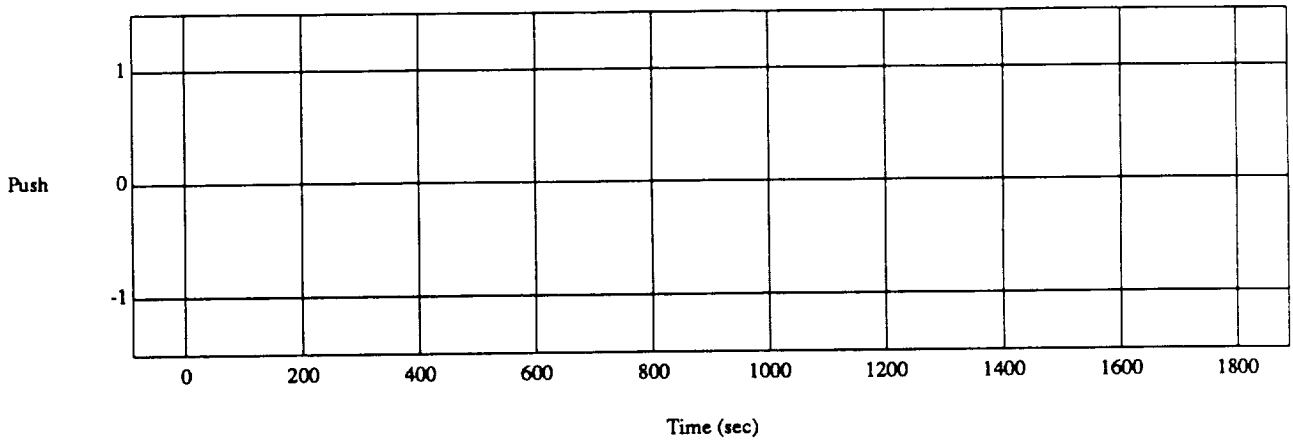


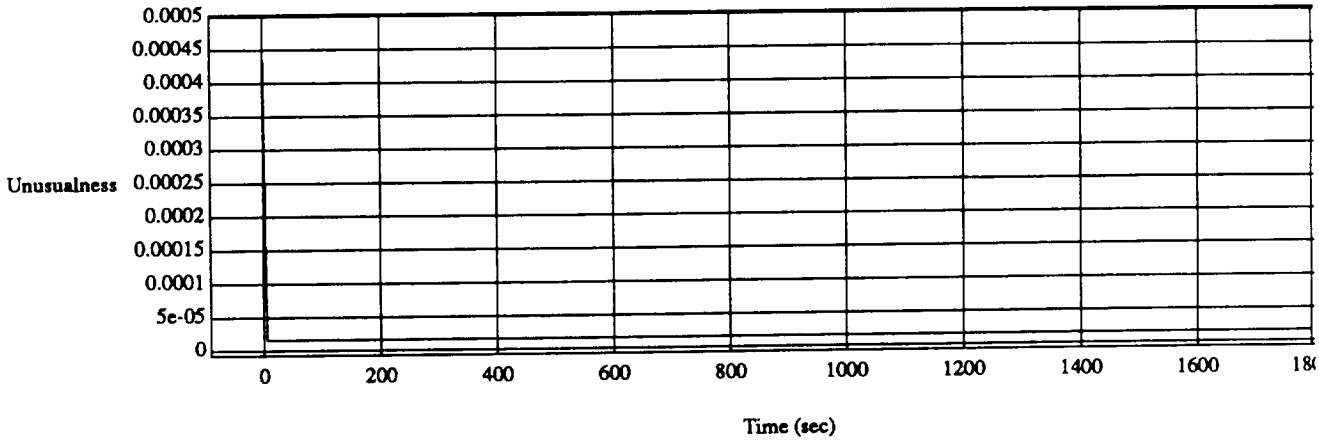


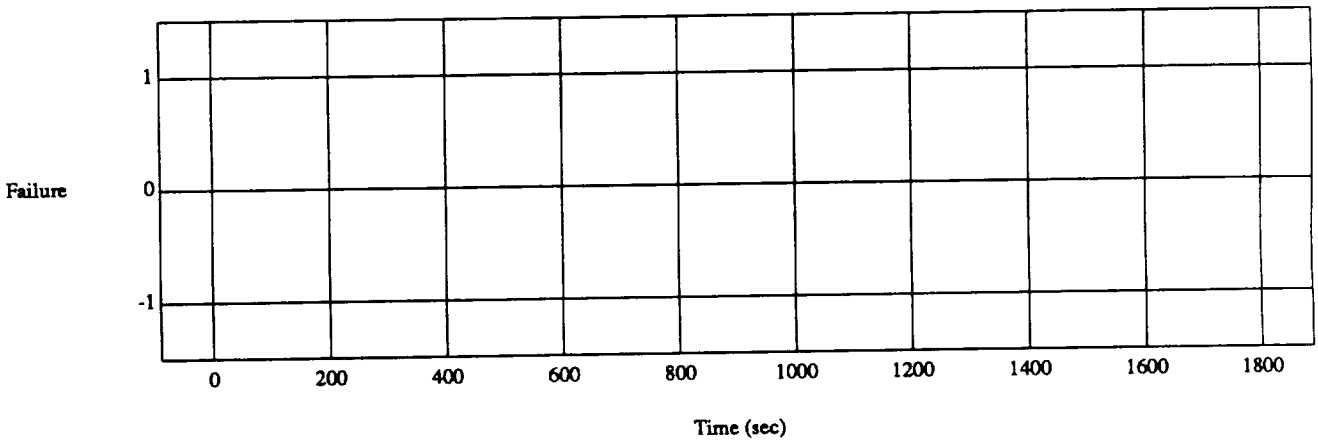


SIMULATION APPLICATION: ARIC Translational Controller Simulation
RUN IDENTIFICATION: Station Keep At 200 Feet
MODEL: ORBITER
DATE: Sat Nov 21 1992 04:20:11 PM
NUMBER OF DATA POINTS: 361
DATA SAMPLING FREQUENCY: 0.200 Hz

Azimuth ARIC Learning Parameters - General Parameters

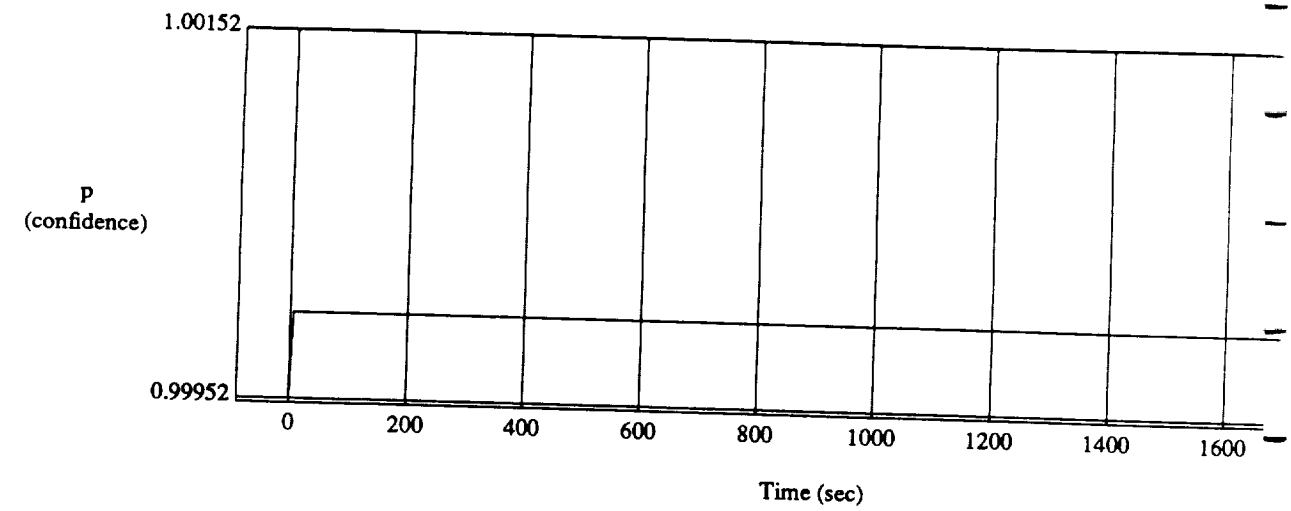


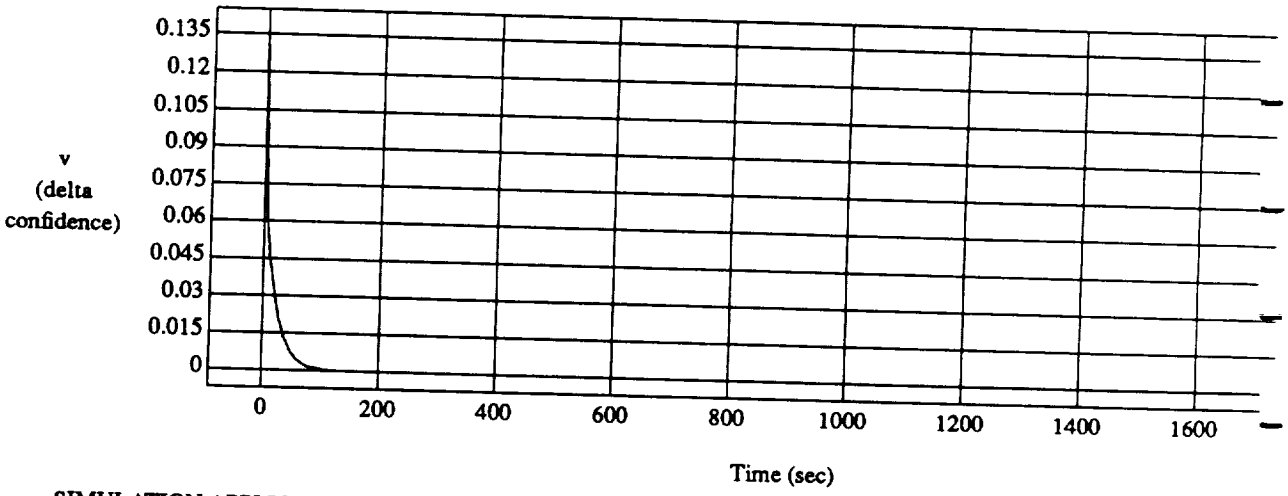




SIMULATION APPLICATION: ARIC Translational Controller Simulation
RUN IDENTIFICATION: Station Keep At 200 Feet
MODEL: ORBITER
DATE: Sat Nov 21 1992 04:20:11 PM
NUMBER OF DATA POINTS: 361
DATA SAMPLING FREQUENCY: 0.200 Hz

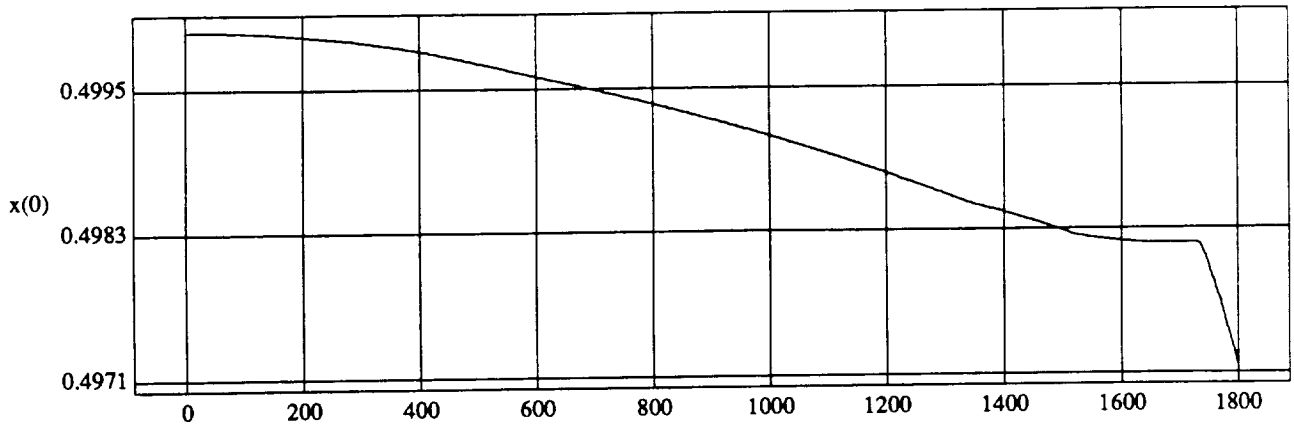
Azimuth ARIC Learning Parameters - Learning Confidence

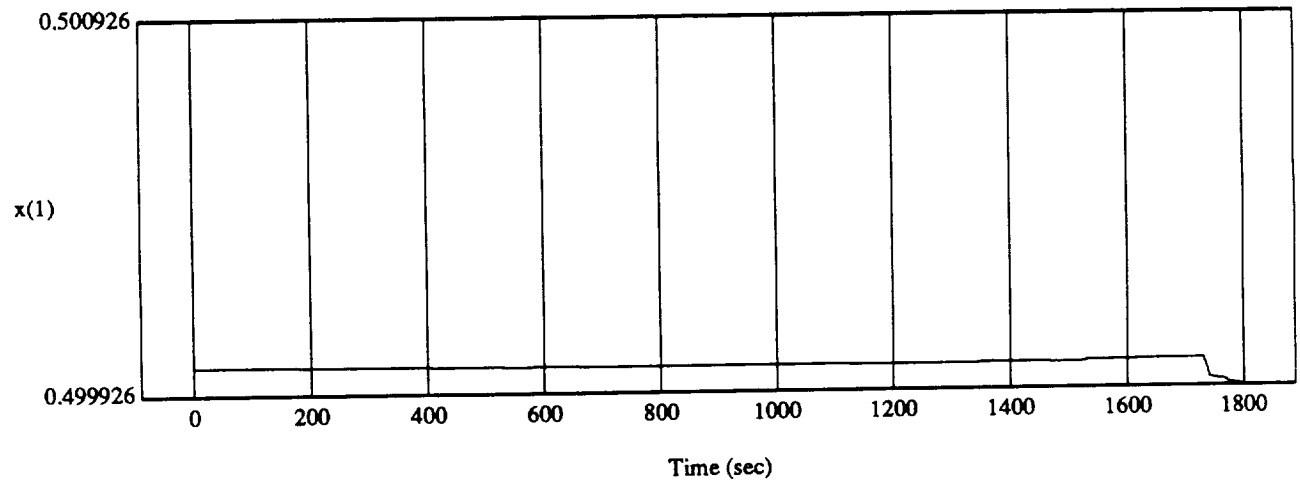




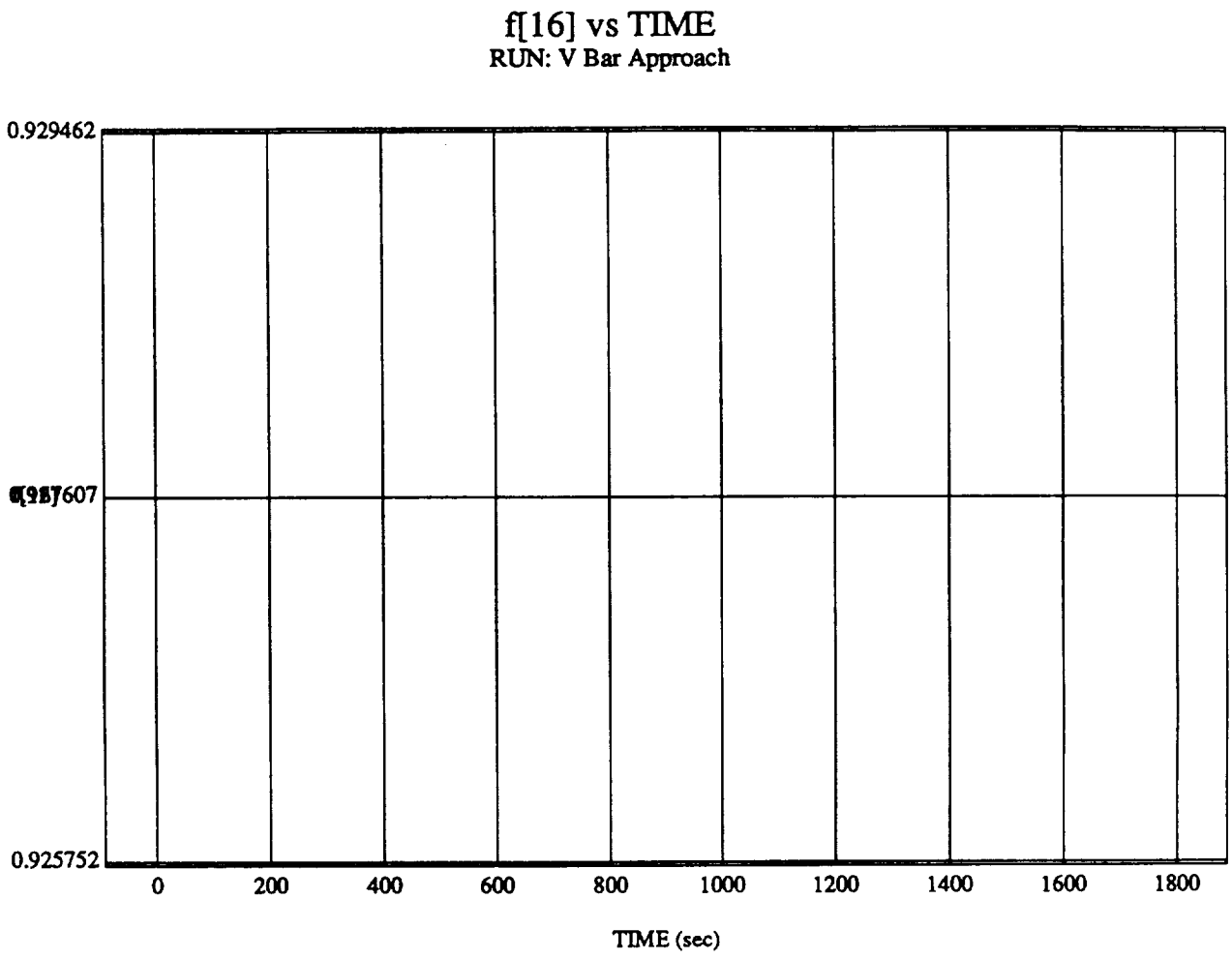
SIMULATION APPLICATION: ARIC Translational Controller Simulation
RUN IDENTIFICATION: Station Keep At 200 Feet
MODEL: ORBITER
DATE: Sat Nov 21 1992 04:20:11 PM
NUMBER OF DATA POINTS: 361
DATA SAMPLING FREQUENCY: 0.200 Hz

Azimuth ARIC Learning Parameters - Scaled Inputs



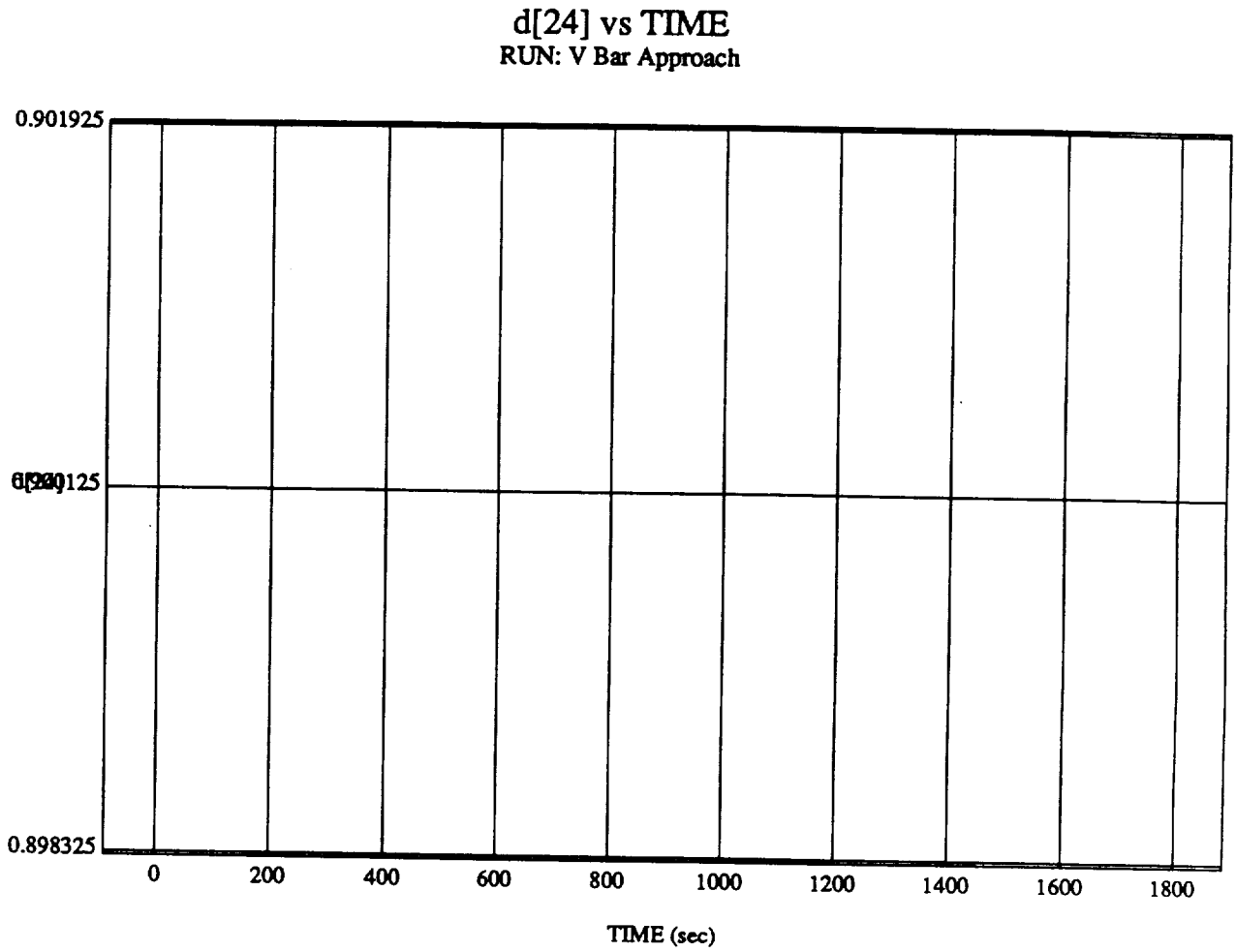


SIMULATION APPLICATION: ARIC Translational Controller Simulation
RUN IDENTIFICATION: Station Keep At 200 Feet
MODEL: ORBITER
DATE: Sat Nov 21 1992 04:20:11 PM
NUMBER OF DATA POINTS: 361
DATA SAMPLING FREQUENCY: 0.200 Hz

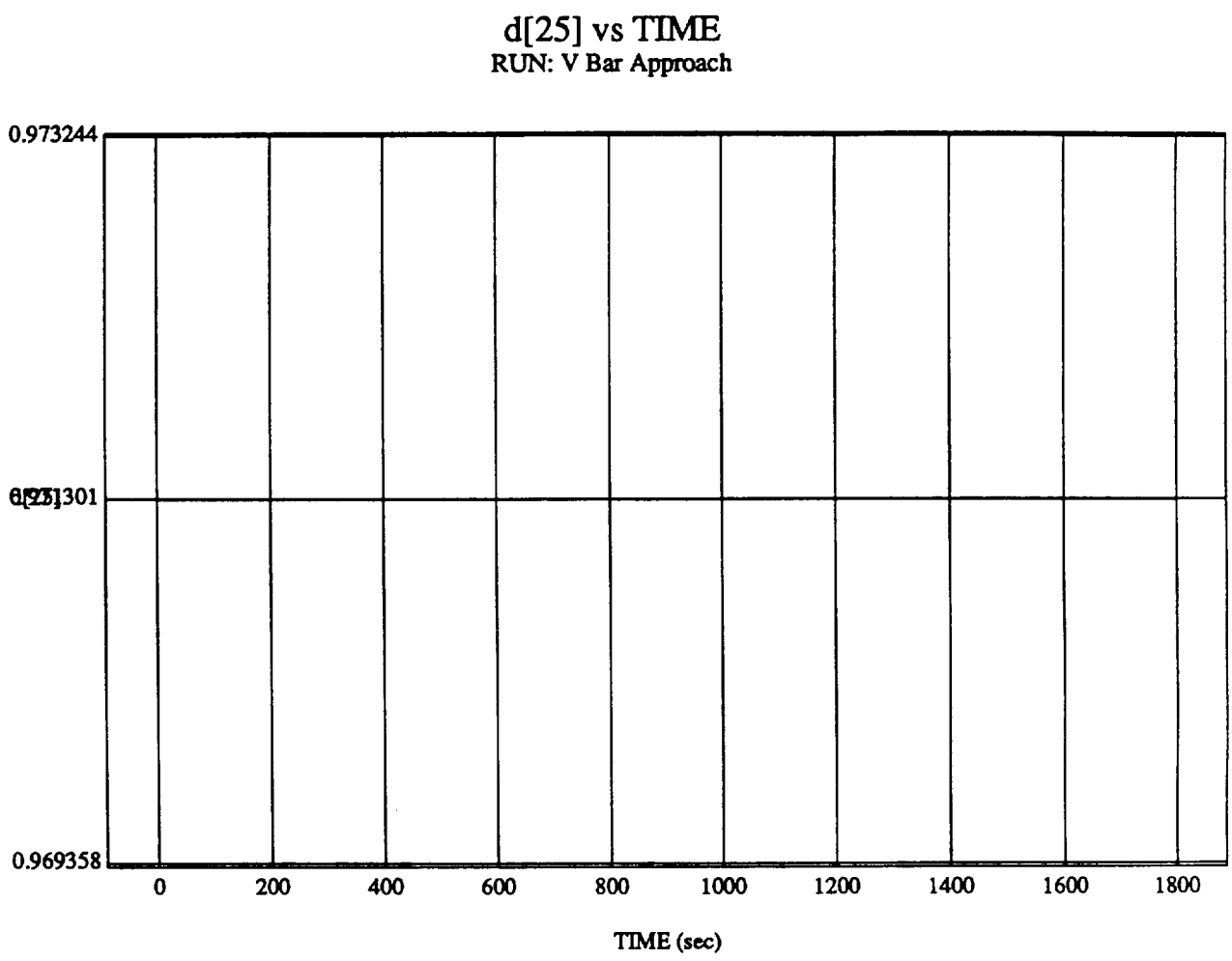


MODULE: ORBITER.lm_range
DATA SAMPLING FREQUENCY: 0.200 Hz

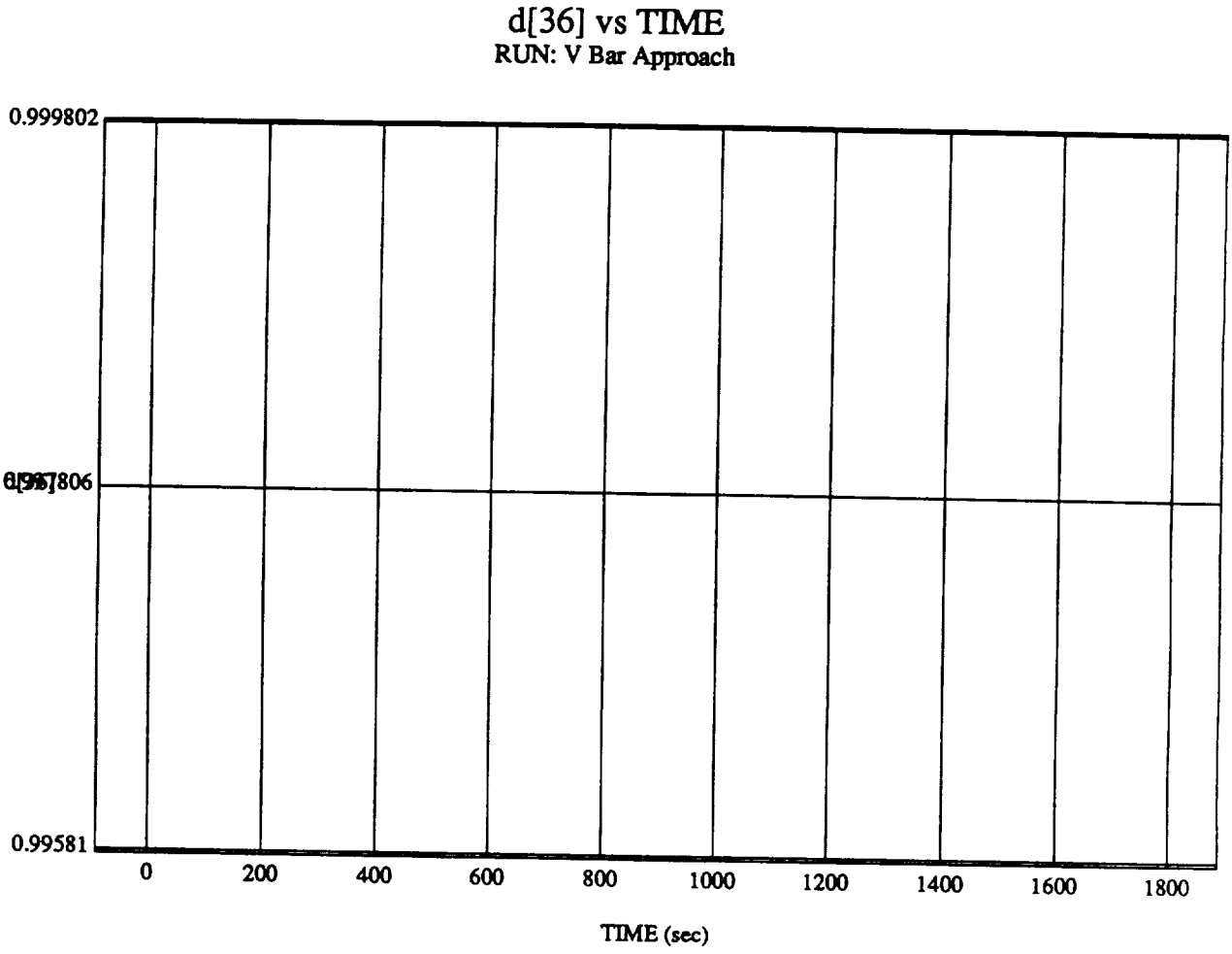
SIMULATION APPLICATION: ARIC Translational Controller Simulation



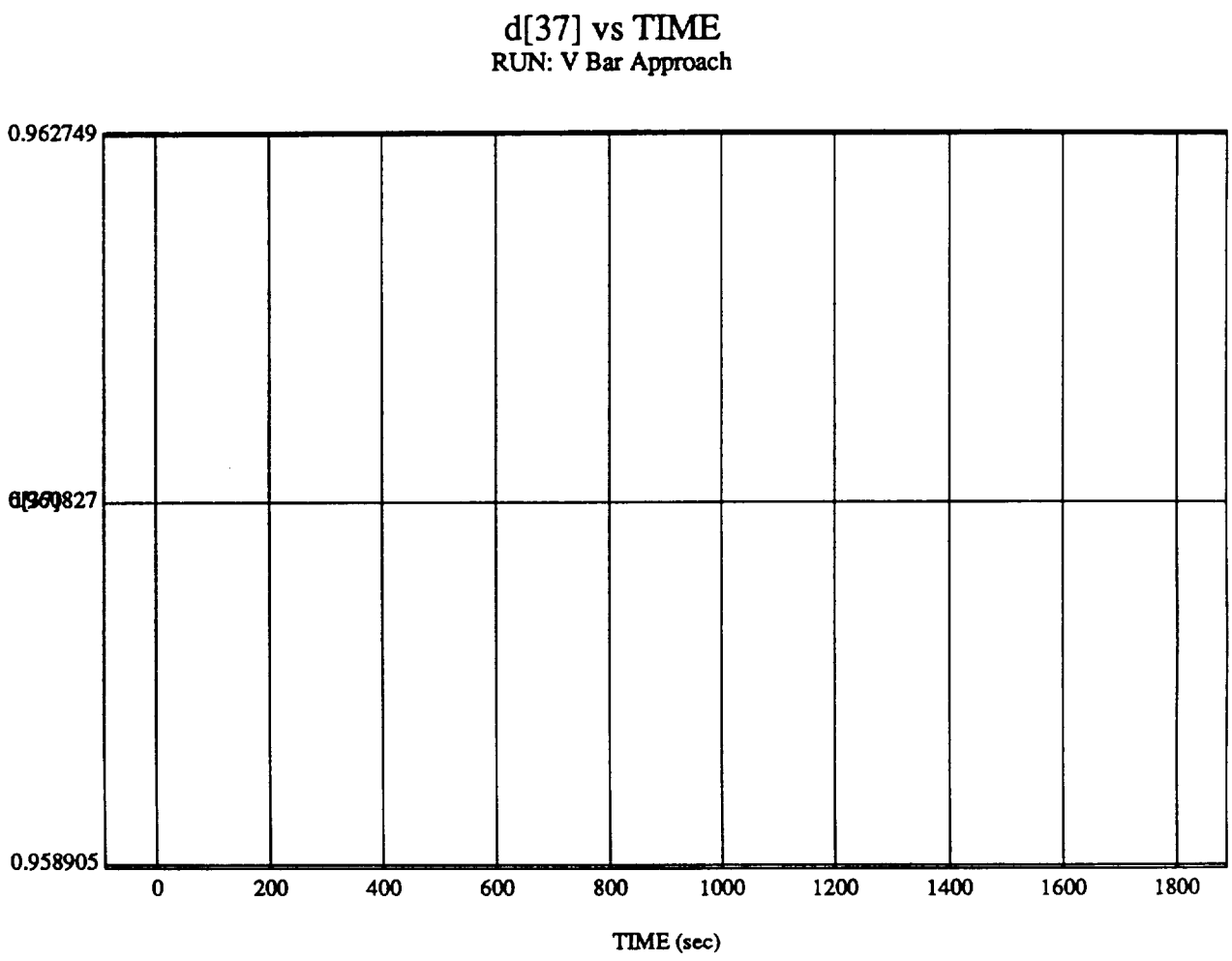
MODULE: ORBITER.lm_range
DATA SAMPLING FREQUENCY: 0.200 Hz



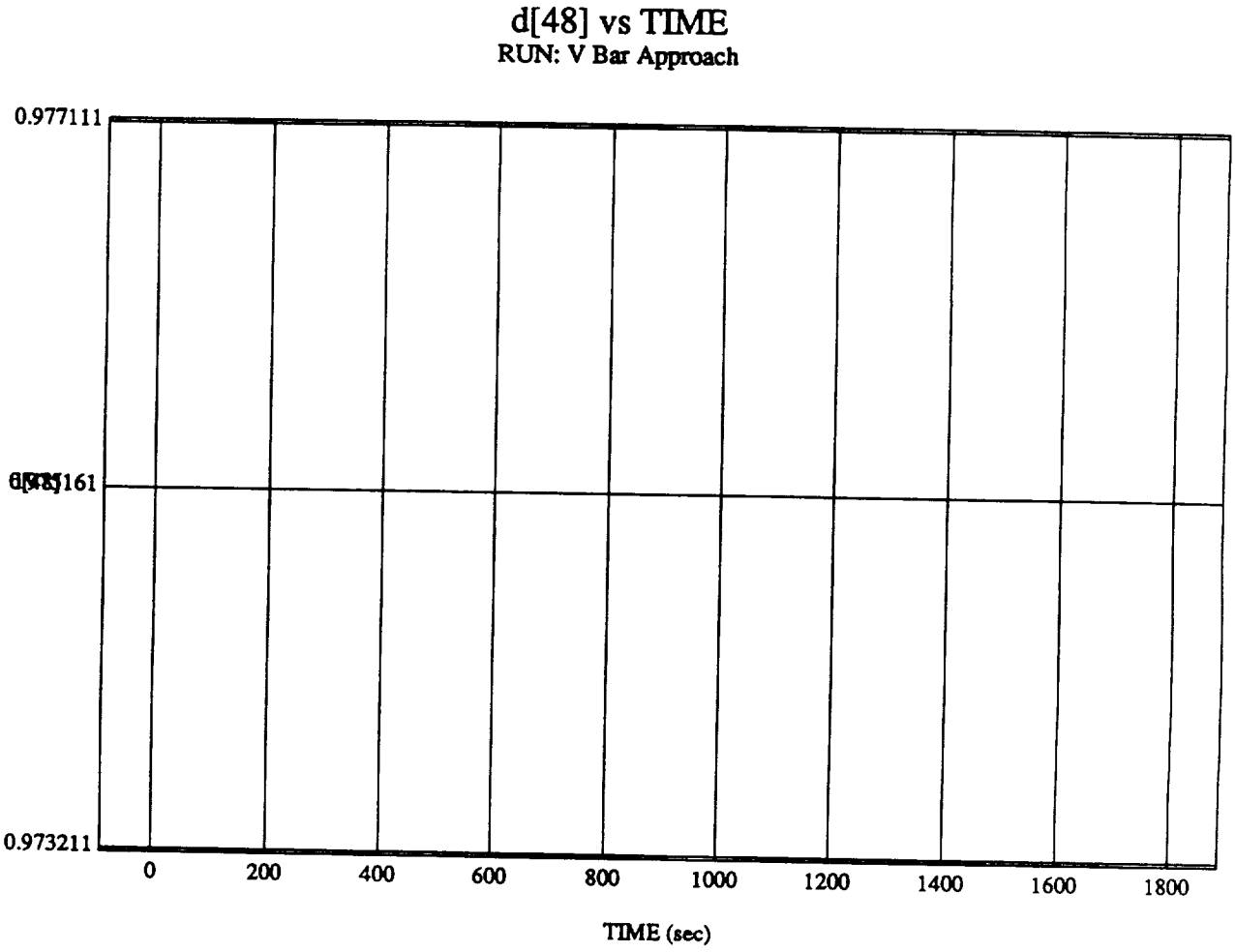
MODULE: ORBITER.lm_range
DATA SAMPLING FREQUENCY: 0.200 Hz



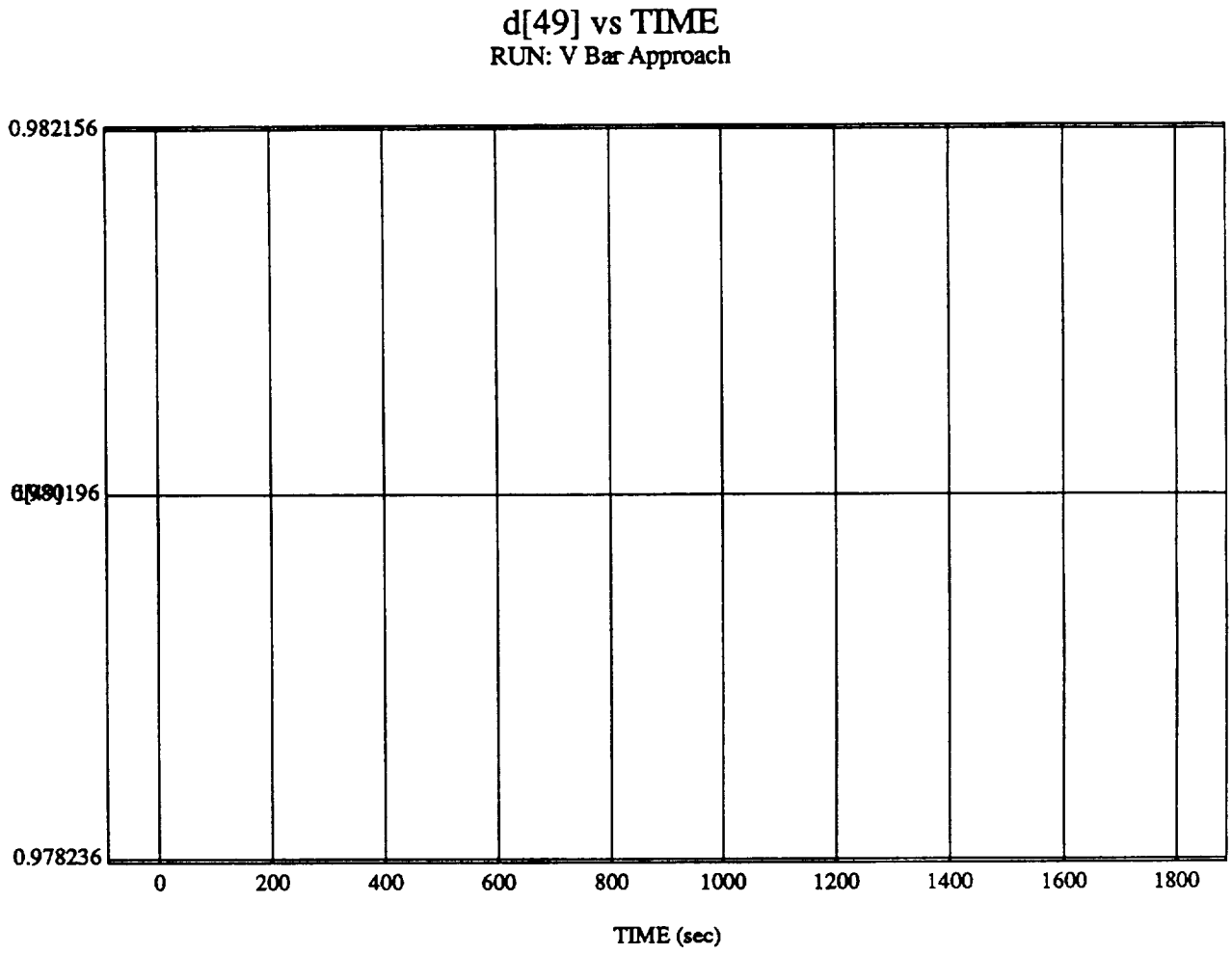
MODULE: ORBITER.lm_range
DATA SAMPLING FREQUENCY: 0.200 Hz



MODULE: ORBITER.lm_range
DATA SAMPLING FREQUENCY: 0.200 Hz



MODULE: ORBITER.lm_range
DATA SAMPLING FREQUENCY: 0.200 Hz



MODULE: ORBITER.lm_range
DATA SAMPLING FREQUENCY: 0.200 Hz

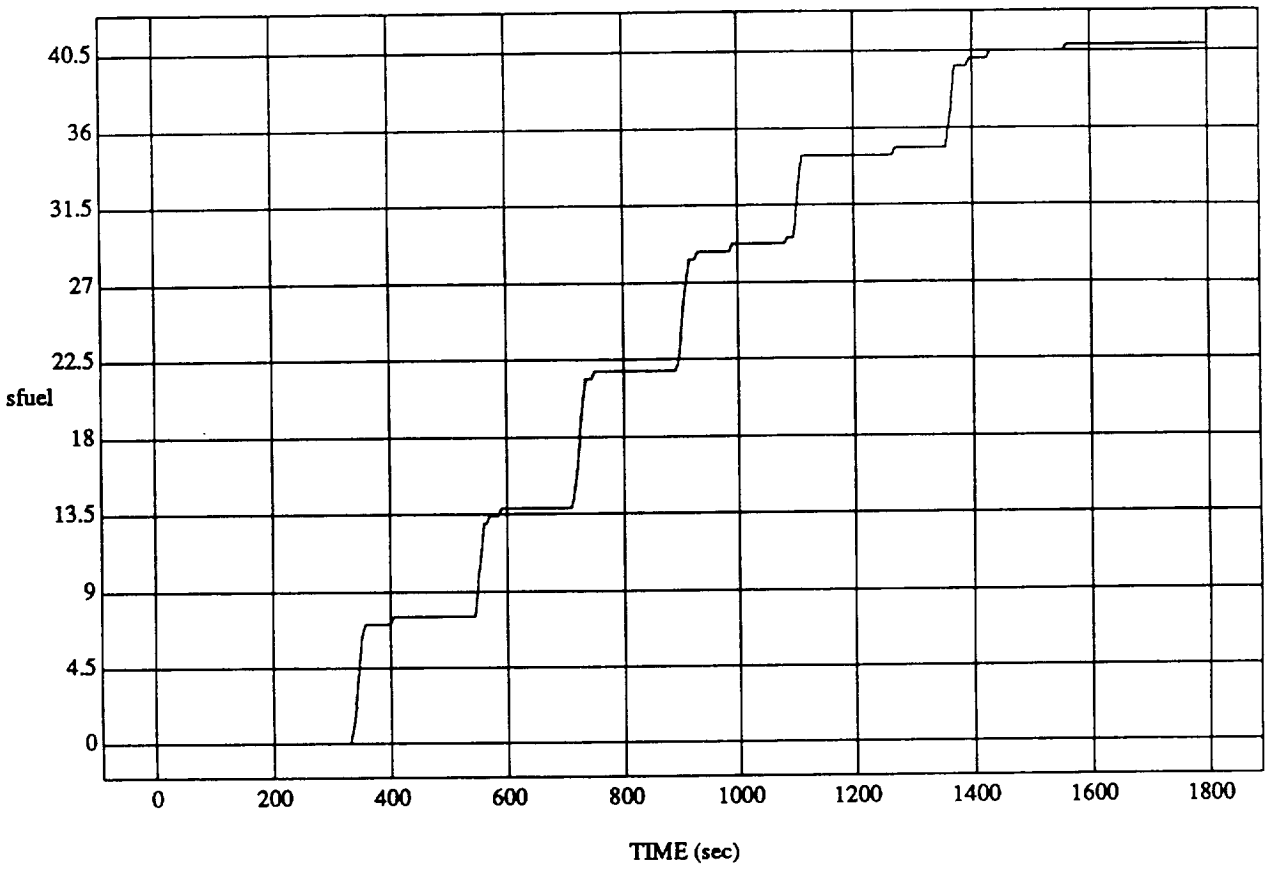
SIMULATION APPLICATION: ARIC Translational Controller Simulation

unusualness vs TIME
RUN: V Bar Approach

SIMULATION APPLICATION: ARIC Translational Controller Simulation

sfuel vs TIME

RUN: V Bar Approach



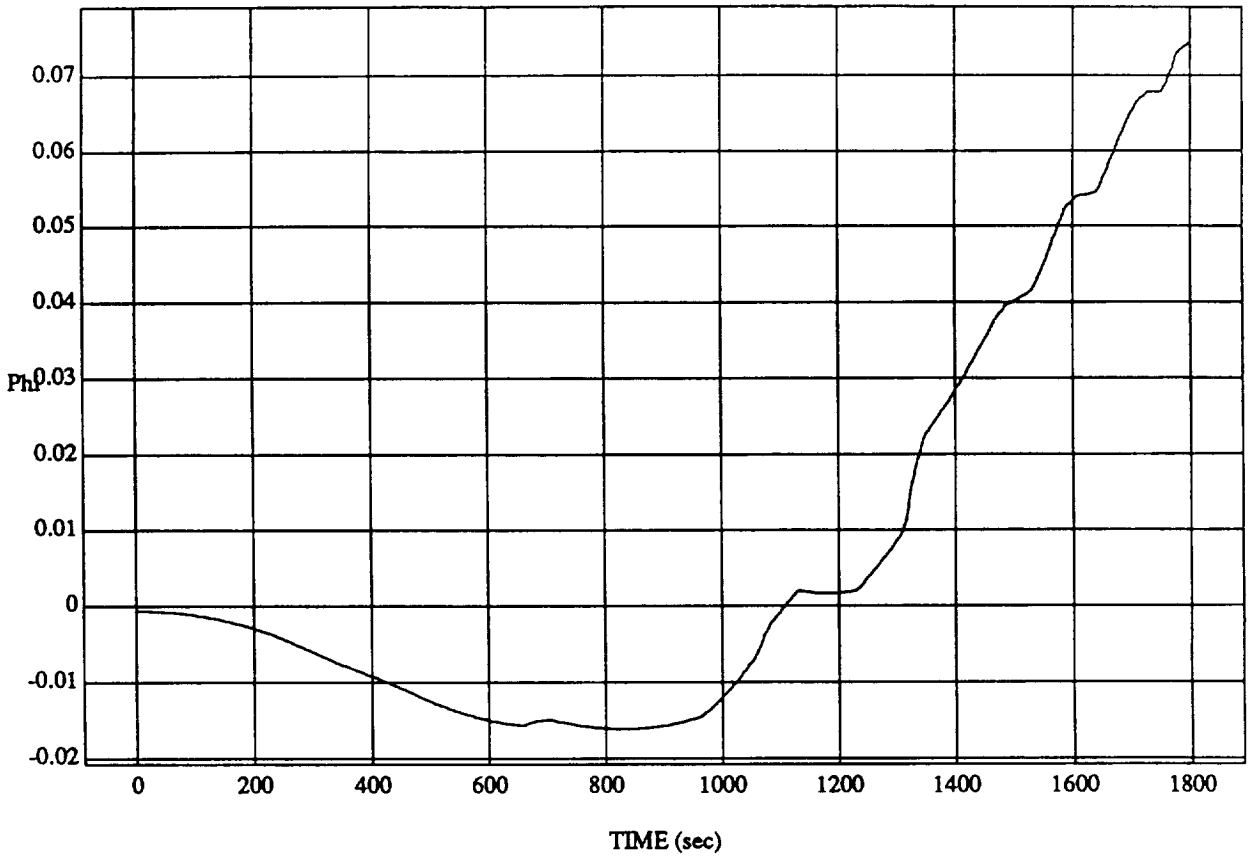
MODULE: ORBITER.primary
DATA SAMPLING FREQUENCY: 0.200 Hz

B2. Shuttle R-bar Approach from 400 feet to 50 feet

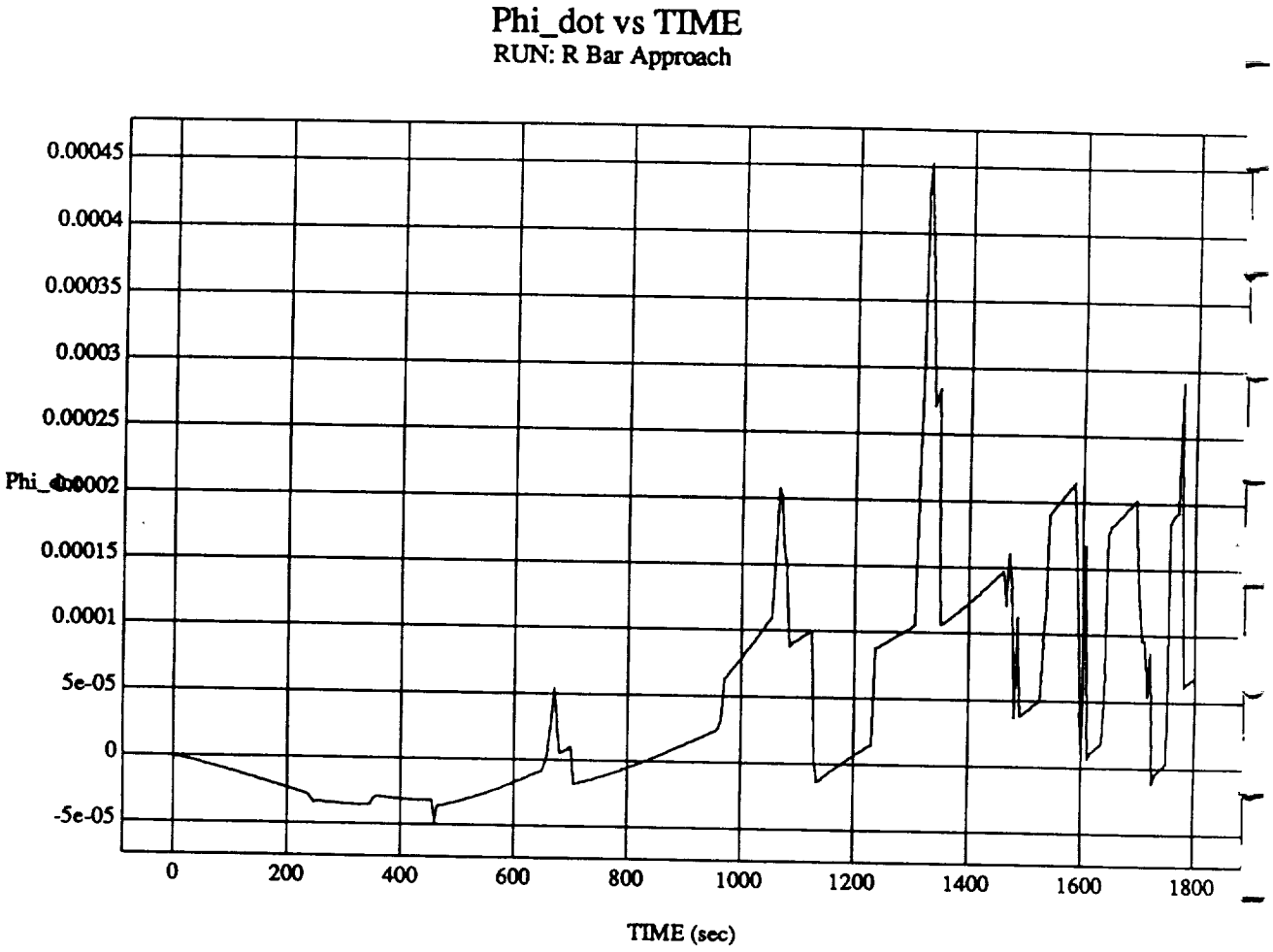
SIMULATION APPLICATION: ARIC Translational Controller Simulation

Phi vs TIME

RUN: R Bar Approach



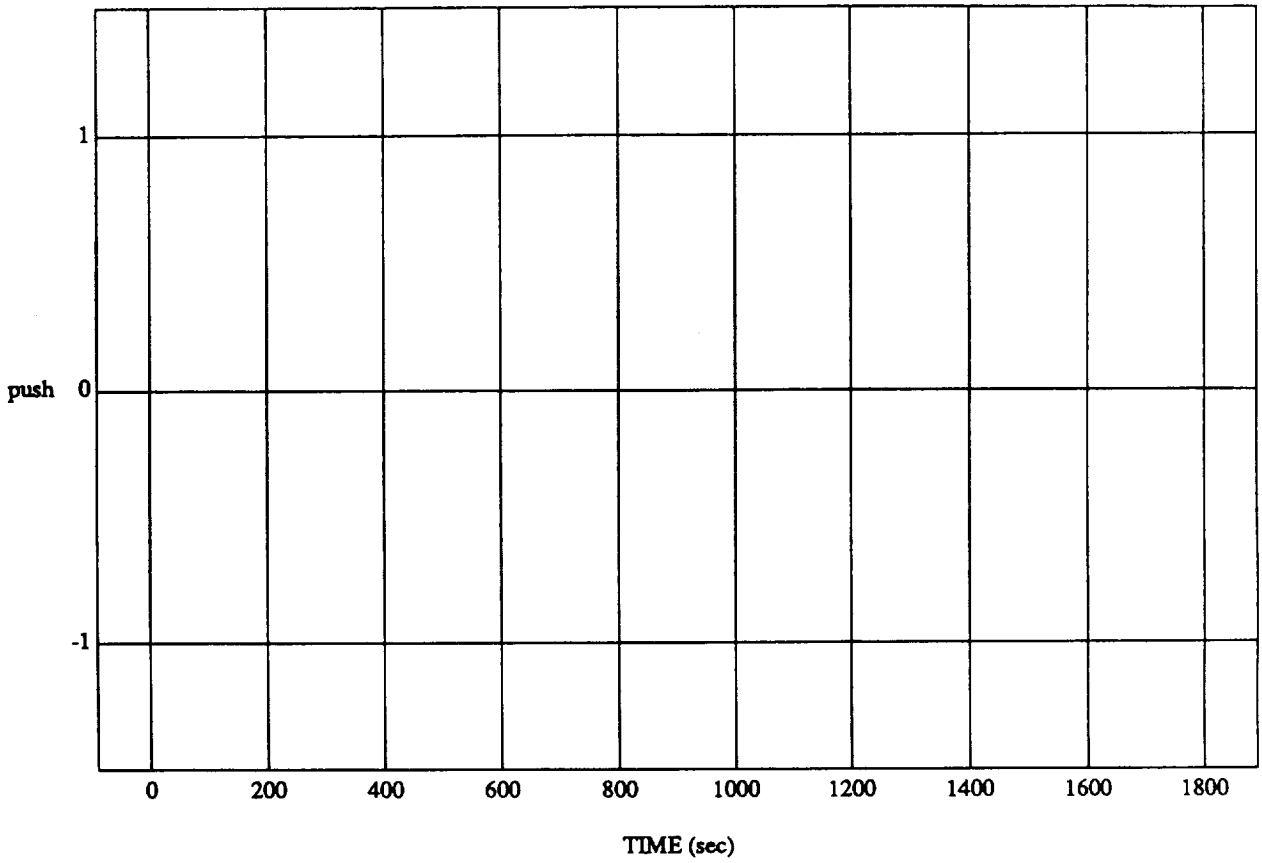
MODULE: ORBITER.lm_azim
DATA SAMPLING FREQUENCY: 0.200 Hz



MODULE: ORBITER.lm_azim
DATA SAMPLING FREQUENCY: 0.200 Hz

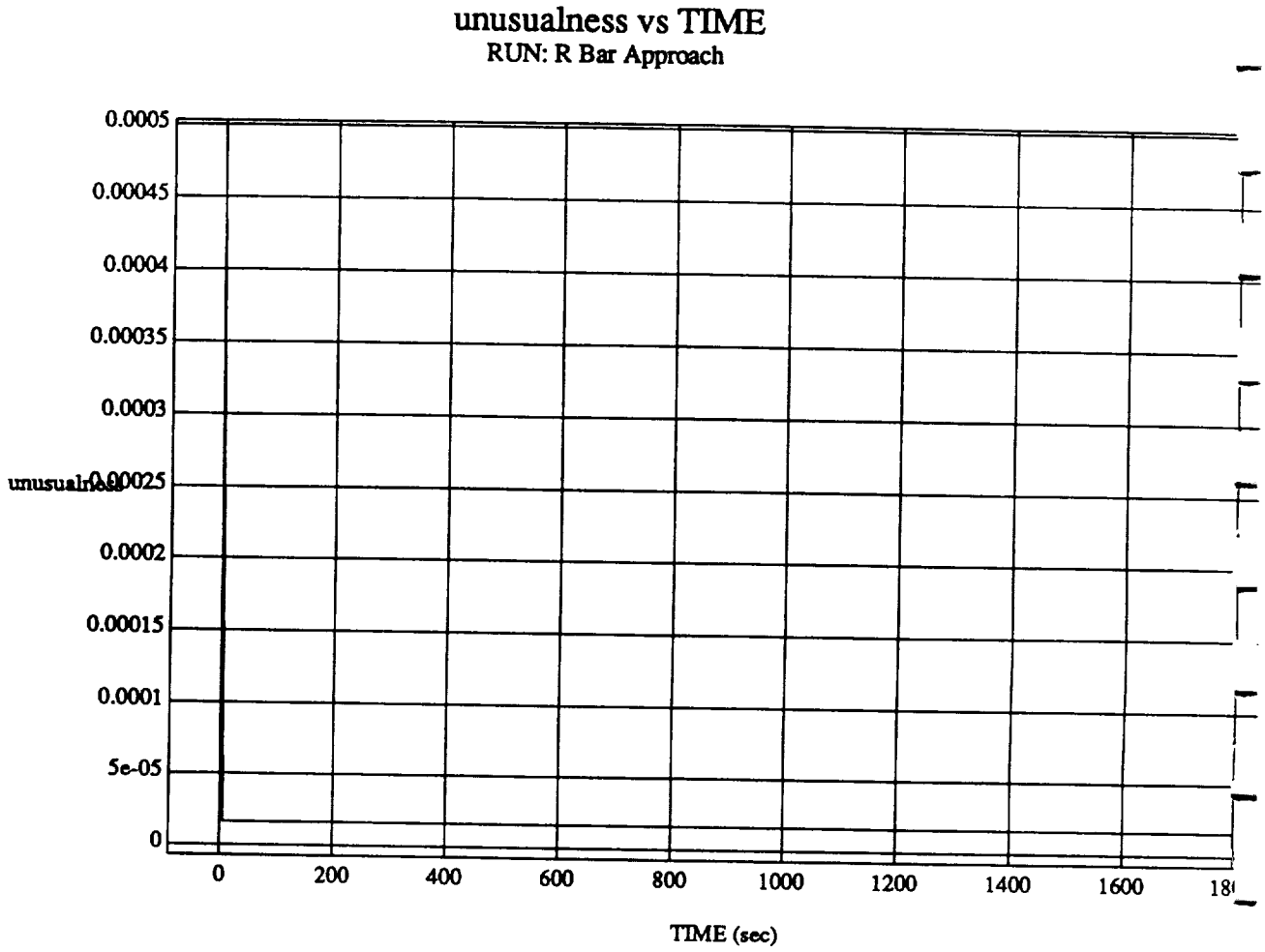
push vs TIME

RUN: R Bar Approach

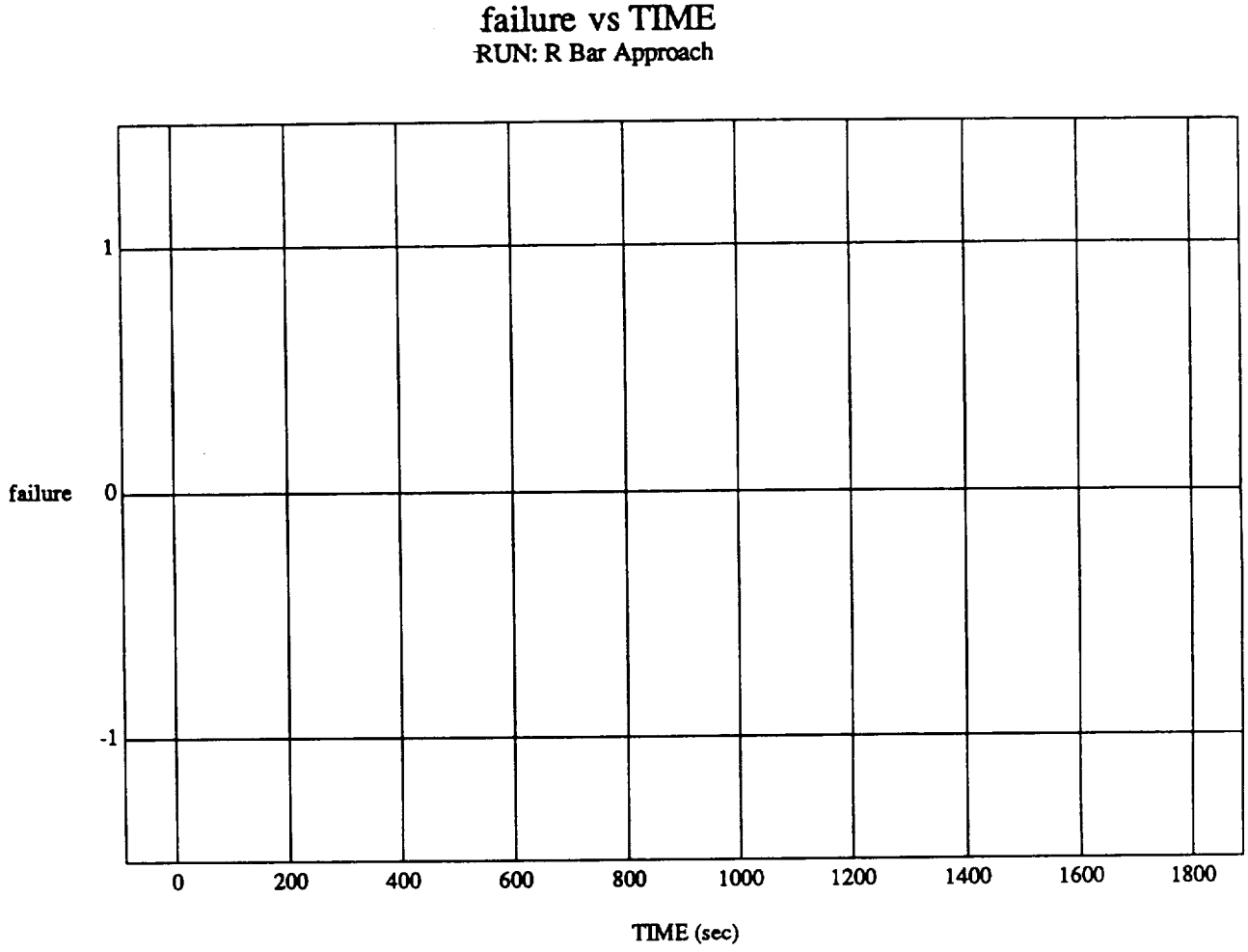


MODULE: ORBITER.lm_azim

DATA SAMPLING FREQUENCY: 0.200 Hz

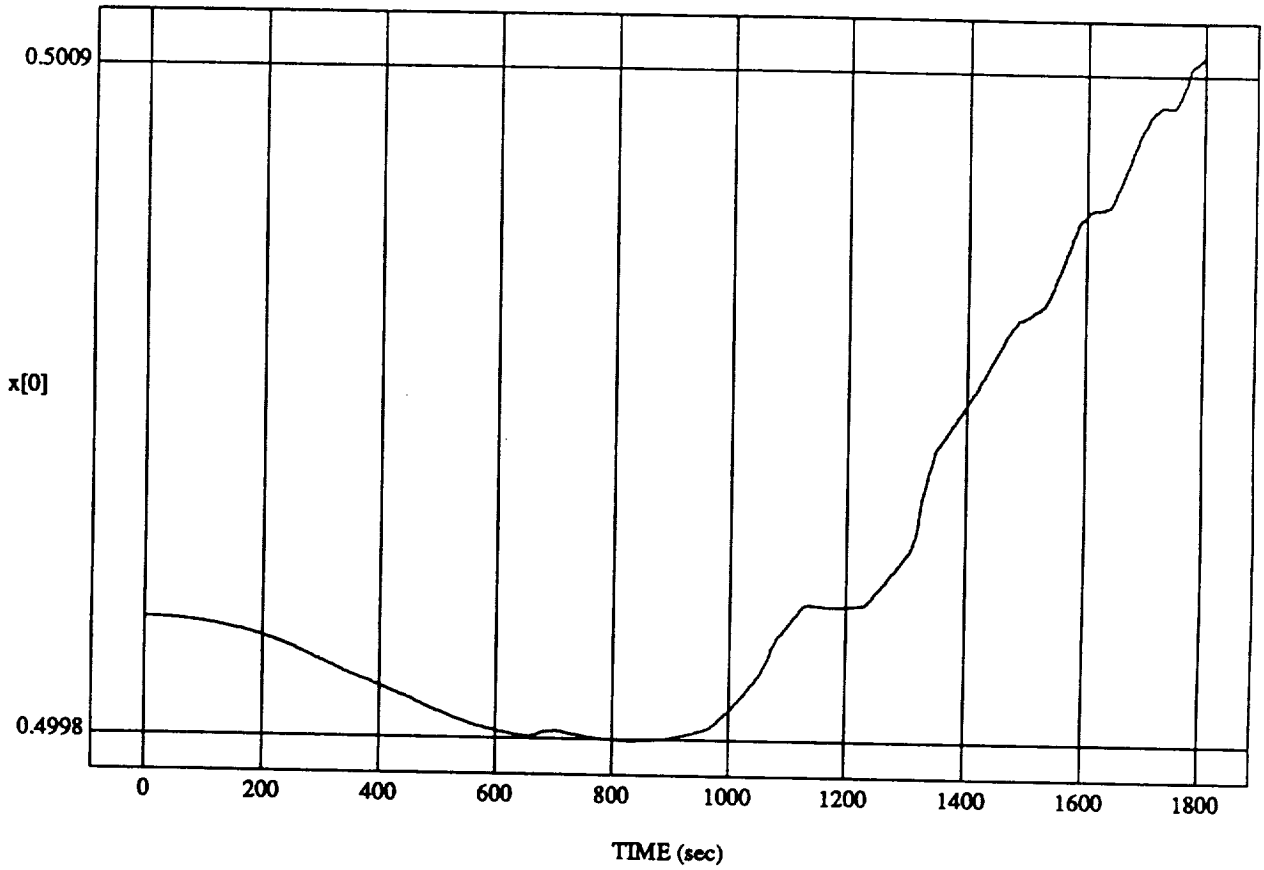


MODULE: ORBITER.lm_azim
DATA SAMPLING FREQUENCY: 0.200 Hz



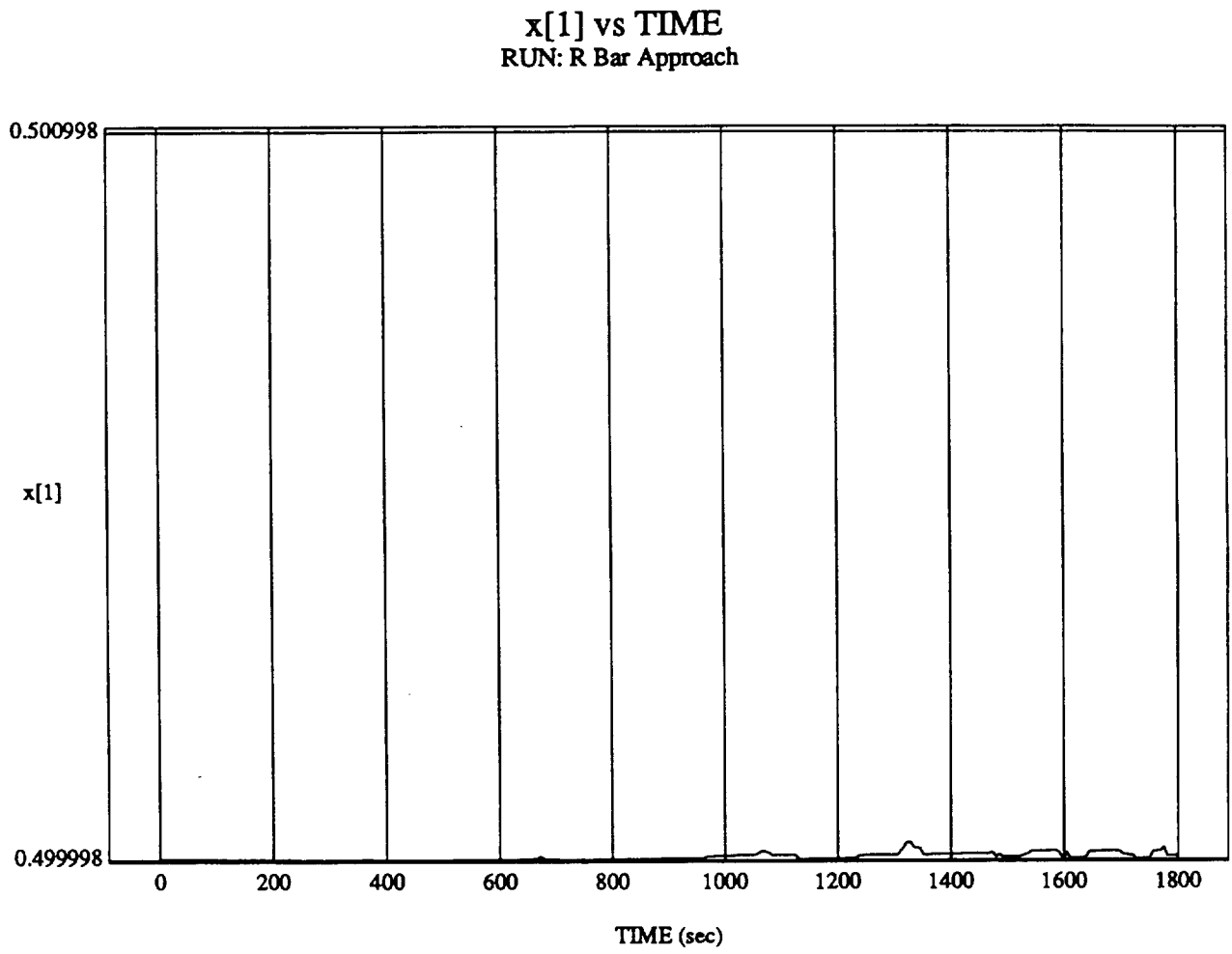
MODULE: ORBITER.lm_azim
DATA SAMPLING FREQUENCY: 0.200 Hz

x[0] vs TIME
RUN: R Bar Approach

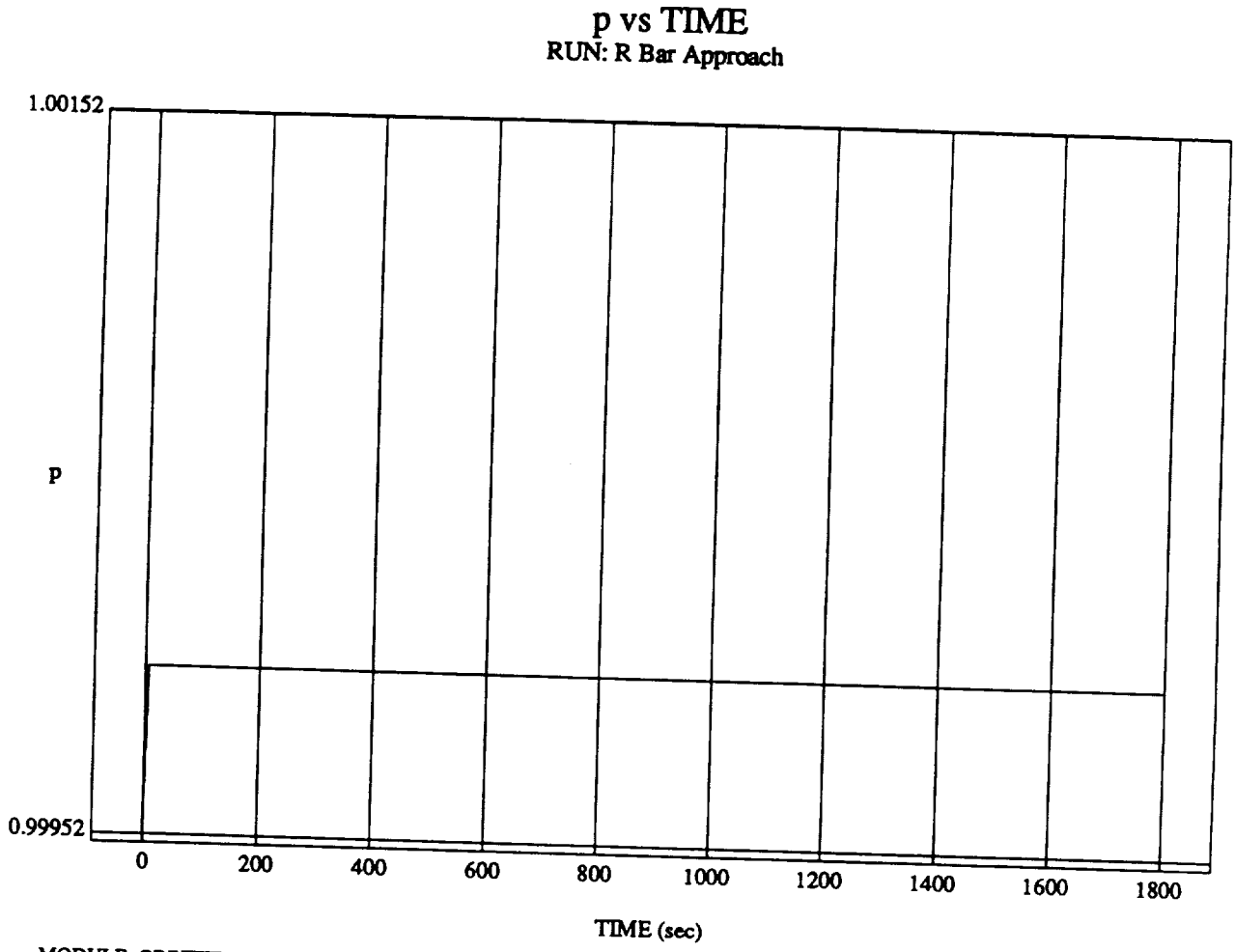


MODULE: ORBITER.lm_azim
DATA SAMPLING FREQUENCY: 0.200 Hz

SIMULATION APPLICATION: ARIC Translational Controller Simulation

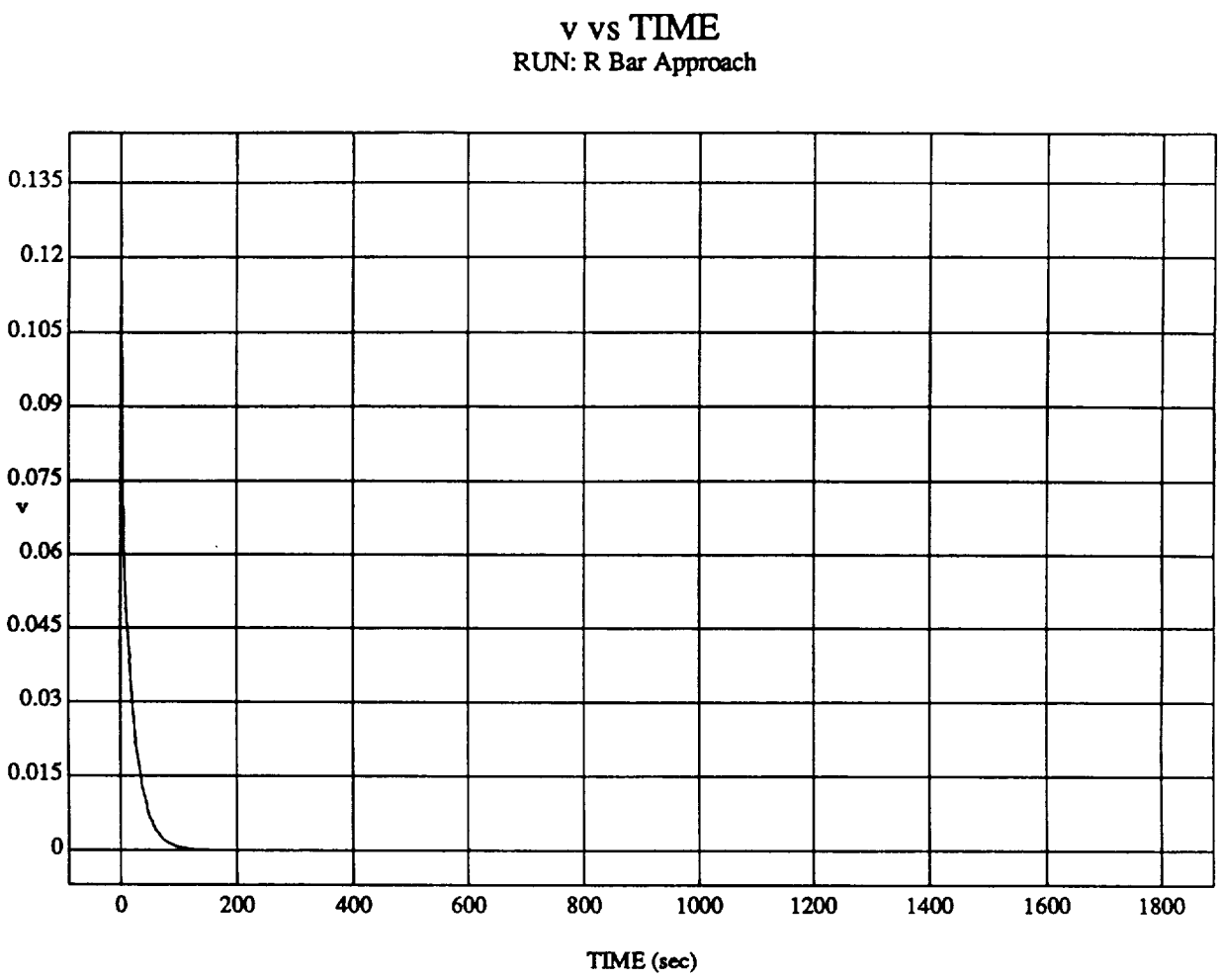


MODULE: ORBITER.lm_azim
DATA SAMPLING FREQUENCY: 0.200 Hz

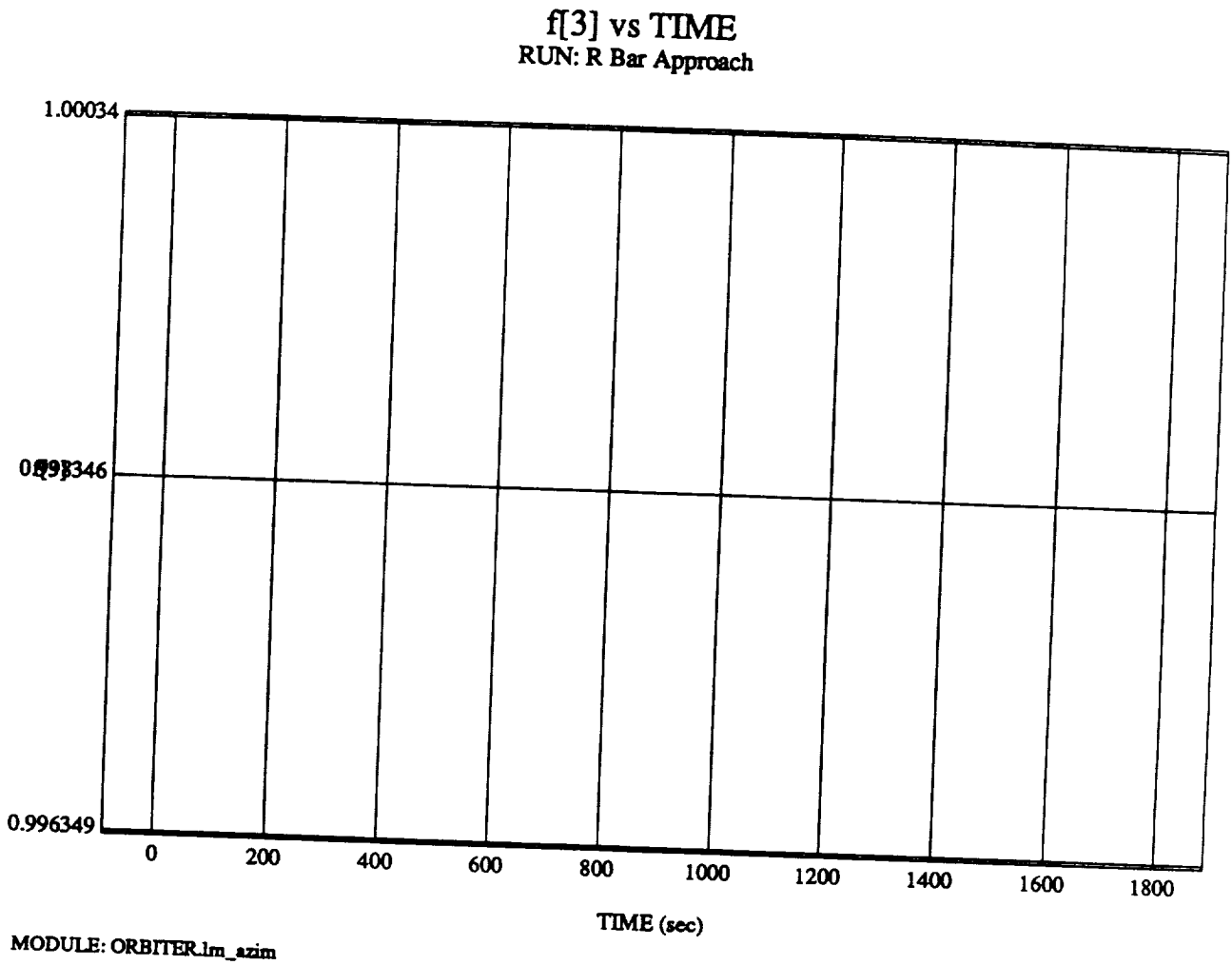


MODULE: ORBITER.lm_azim
DATA SAMPLING FREQUENCY: 0.200 Hz

SIMULATION APPLICATION: ARIC Translational Controller Simulation

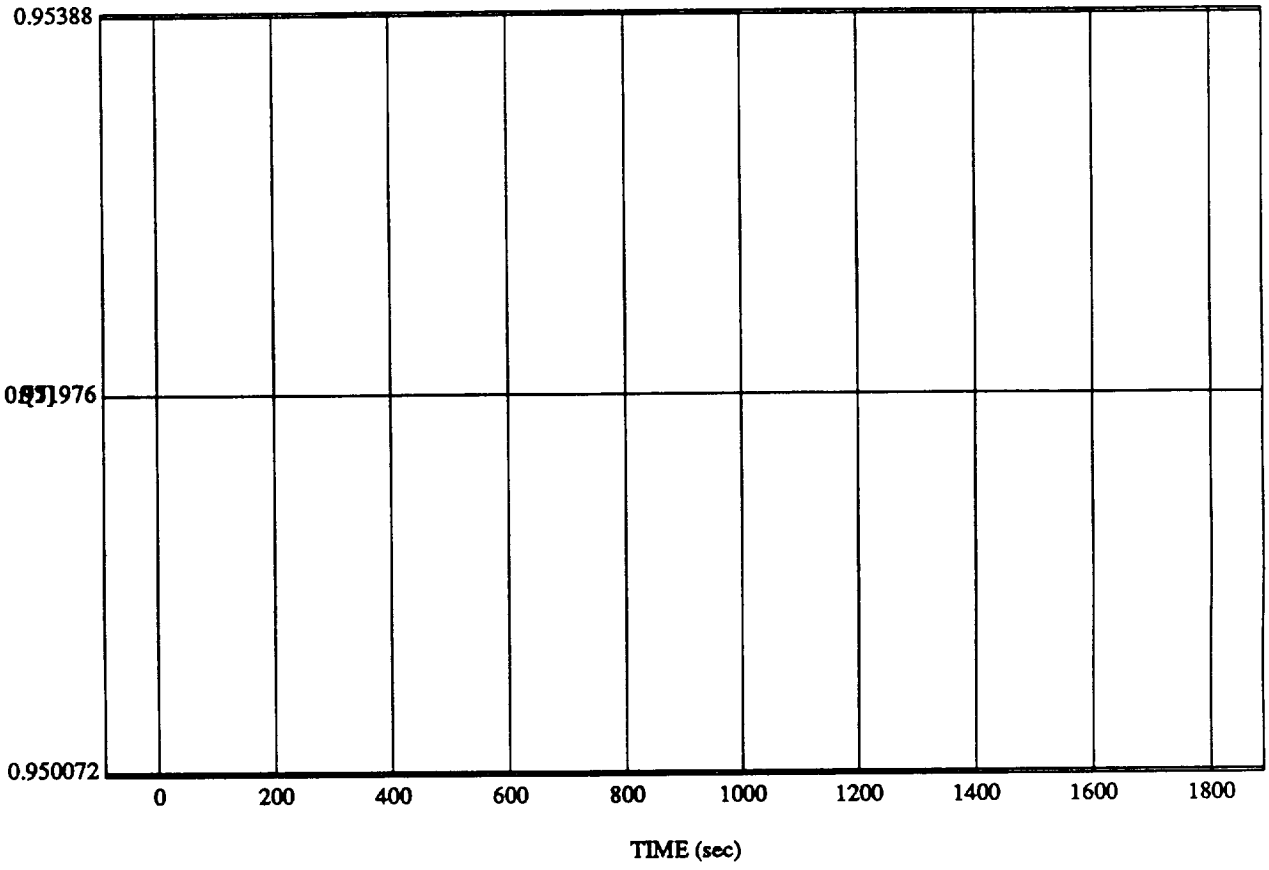


MODULE: ORBITER.lm_azim
DATA SAMPLING FREQUENCY: 0.200 Hz



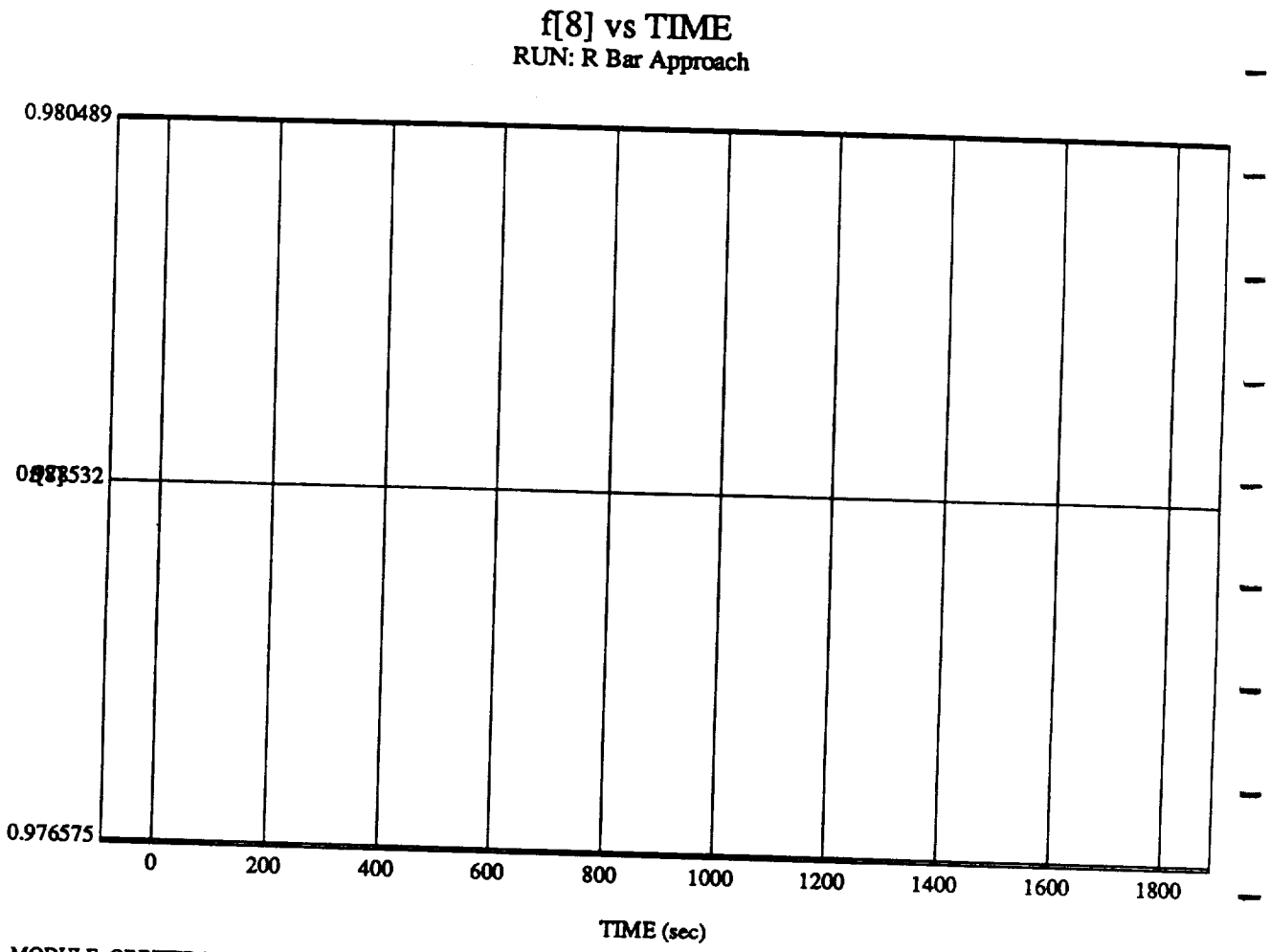
MODULE: ORBITER.lm_azim
DATA SAMPLING FREQUENCY: 0.200 Hz

f[7] vs TIME
RUN: R Bar Approach



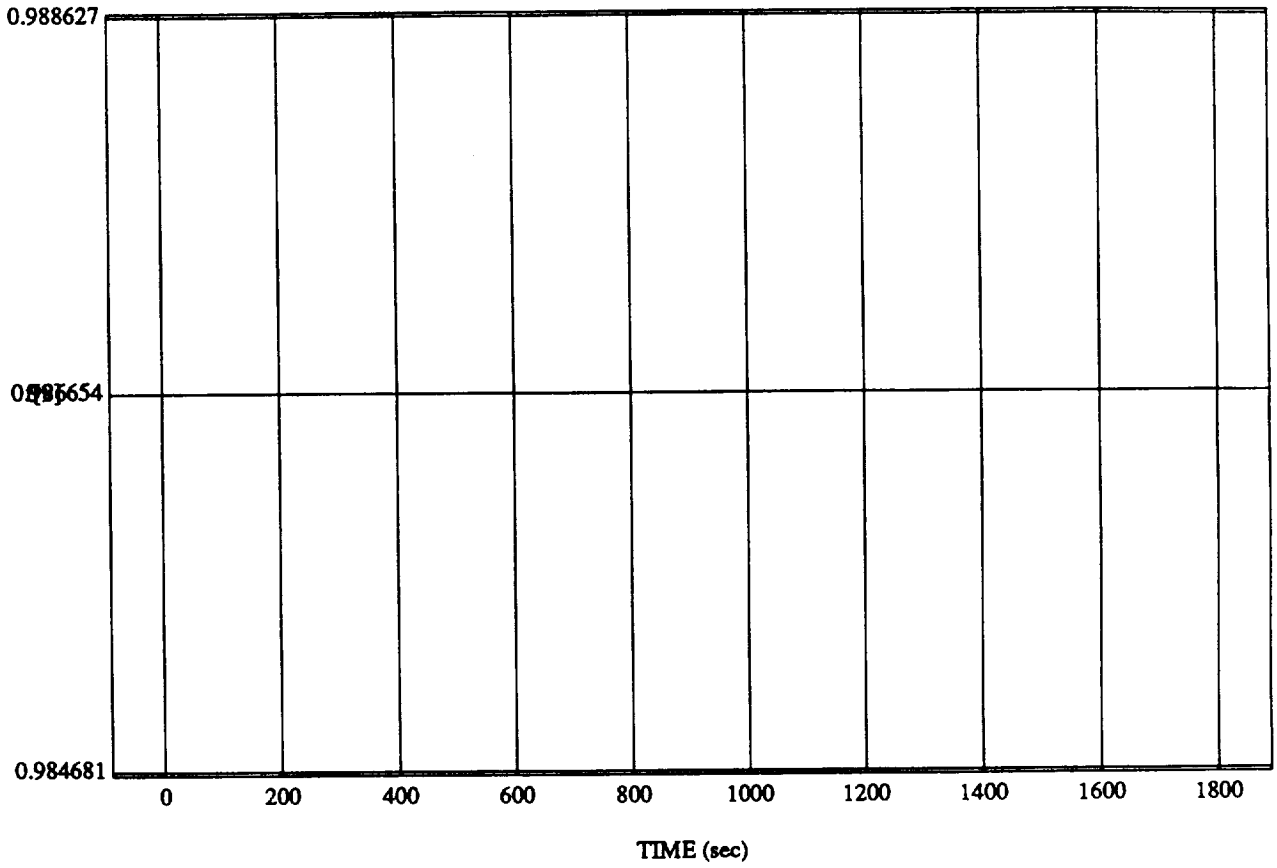
MODULE: ORBITER.lm_azim
DATA SAMPLING FREQUENCY: 0.200 Hz

SIMULATION APPLICATION: ARIC Translational Controller Simulation

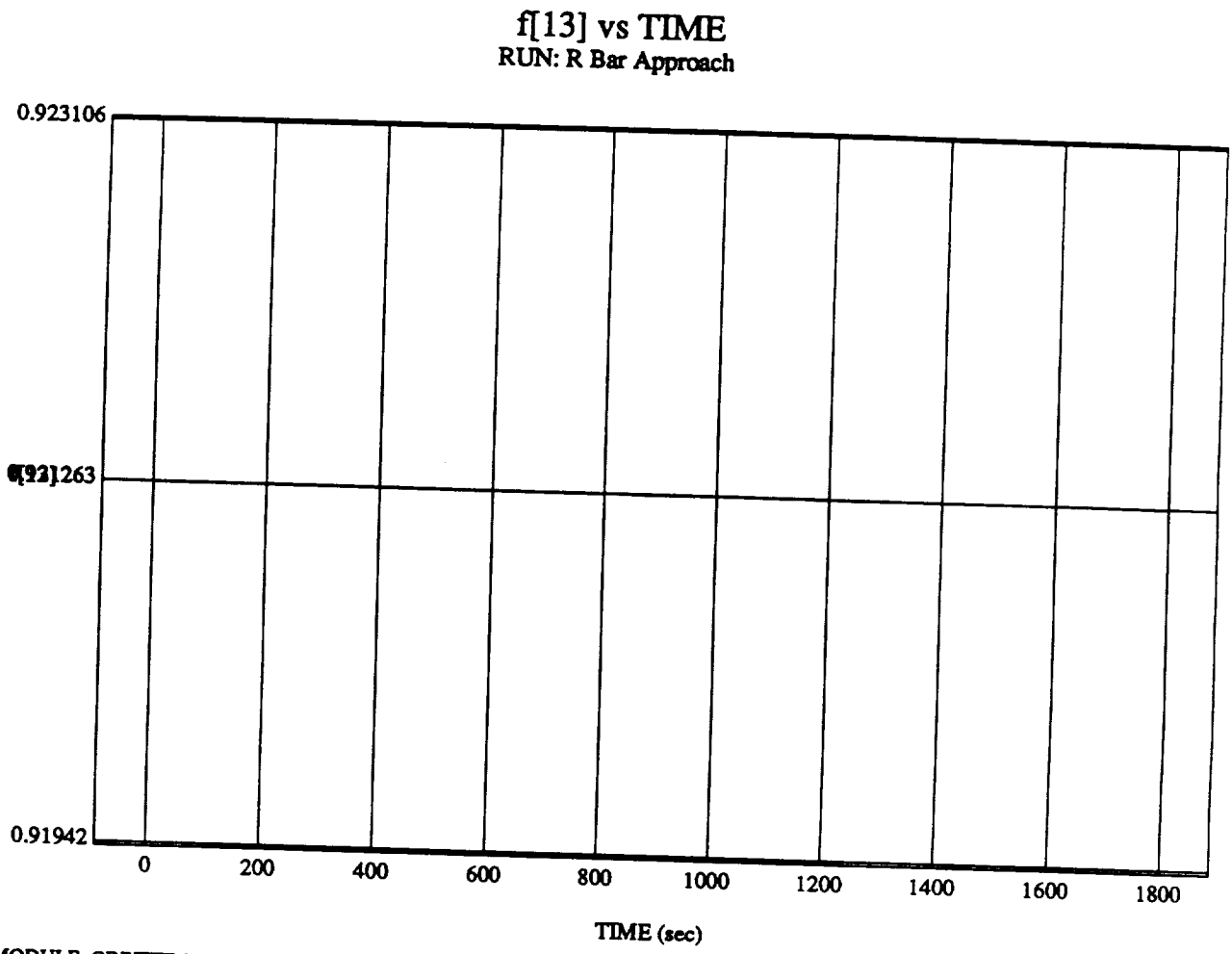


MODULE: ORBITER.lm_azim
DATA SAMPLING FREQUENCY: 0.200 Hz

f[9] vs TIME
RUN: R Bar Approach

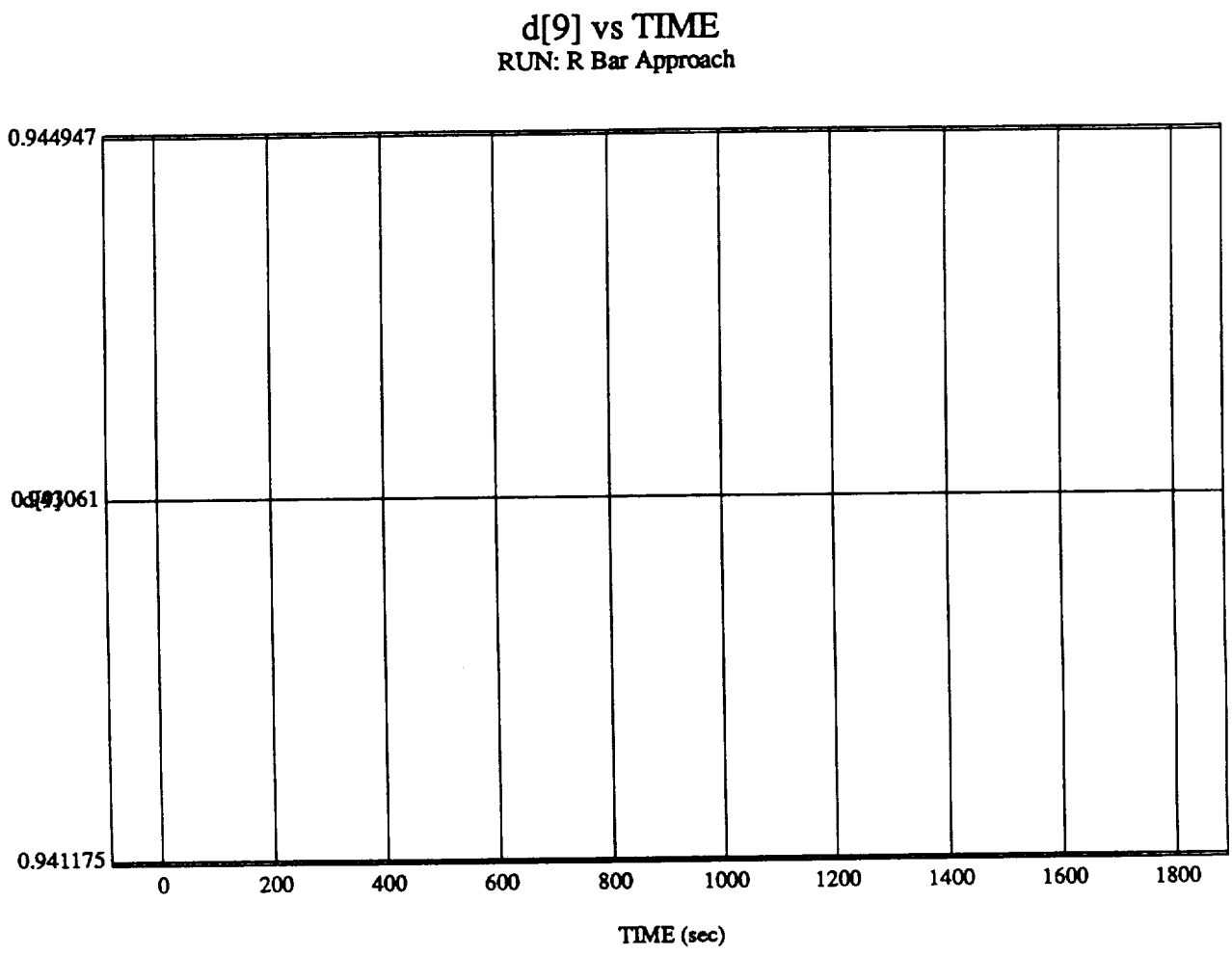


MODULE: ORBITER.lm_azim
DATA SAMPLING FREQUENCY: 0.200 Hz

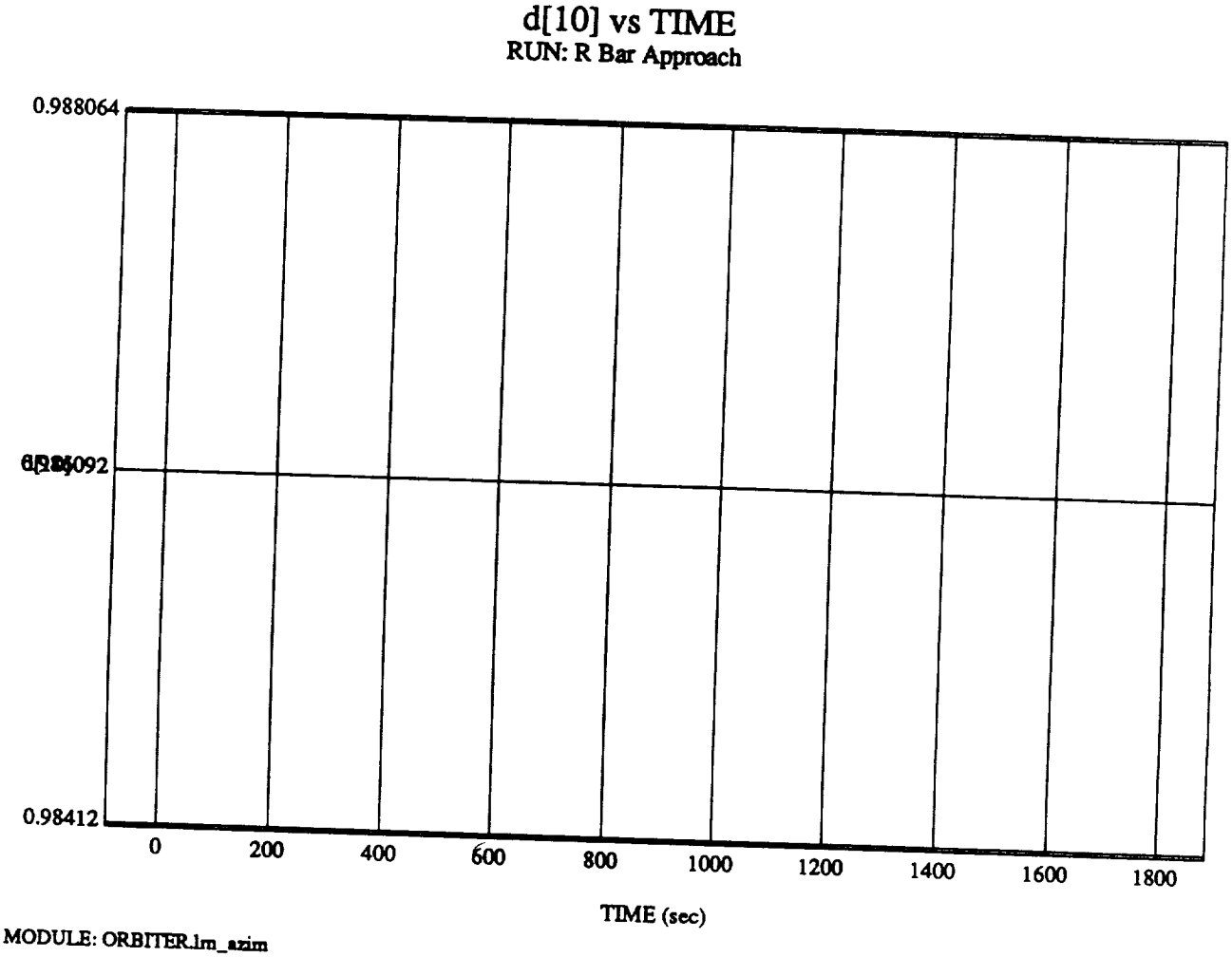


MODULE: ORBITER.lm_azim
DATA SAMPLING FREQUENCY: 0.200 Hz

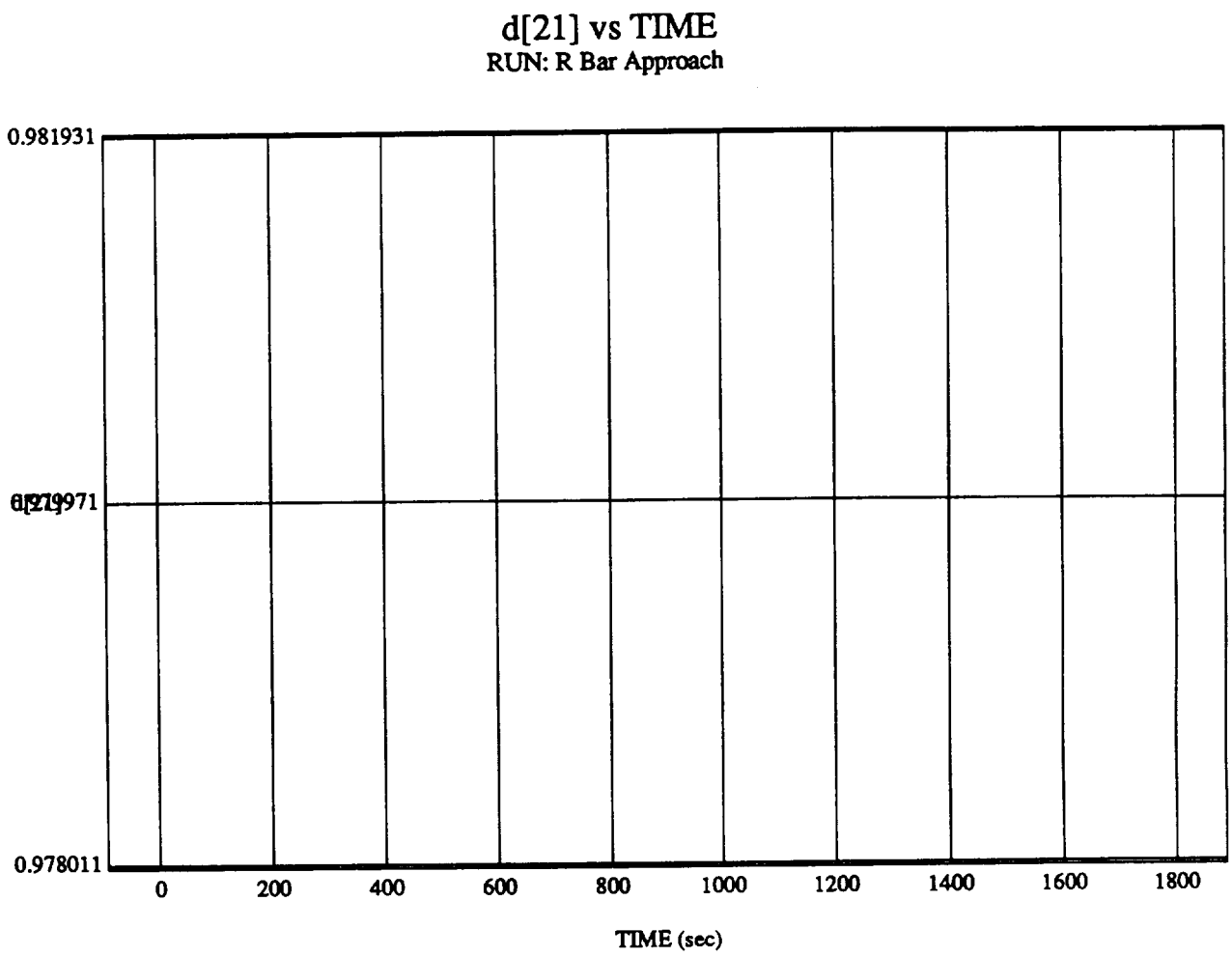
SIMULATION APPLICATION: ARIC Translational Controller Simulation



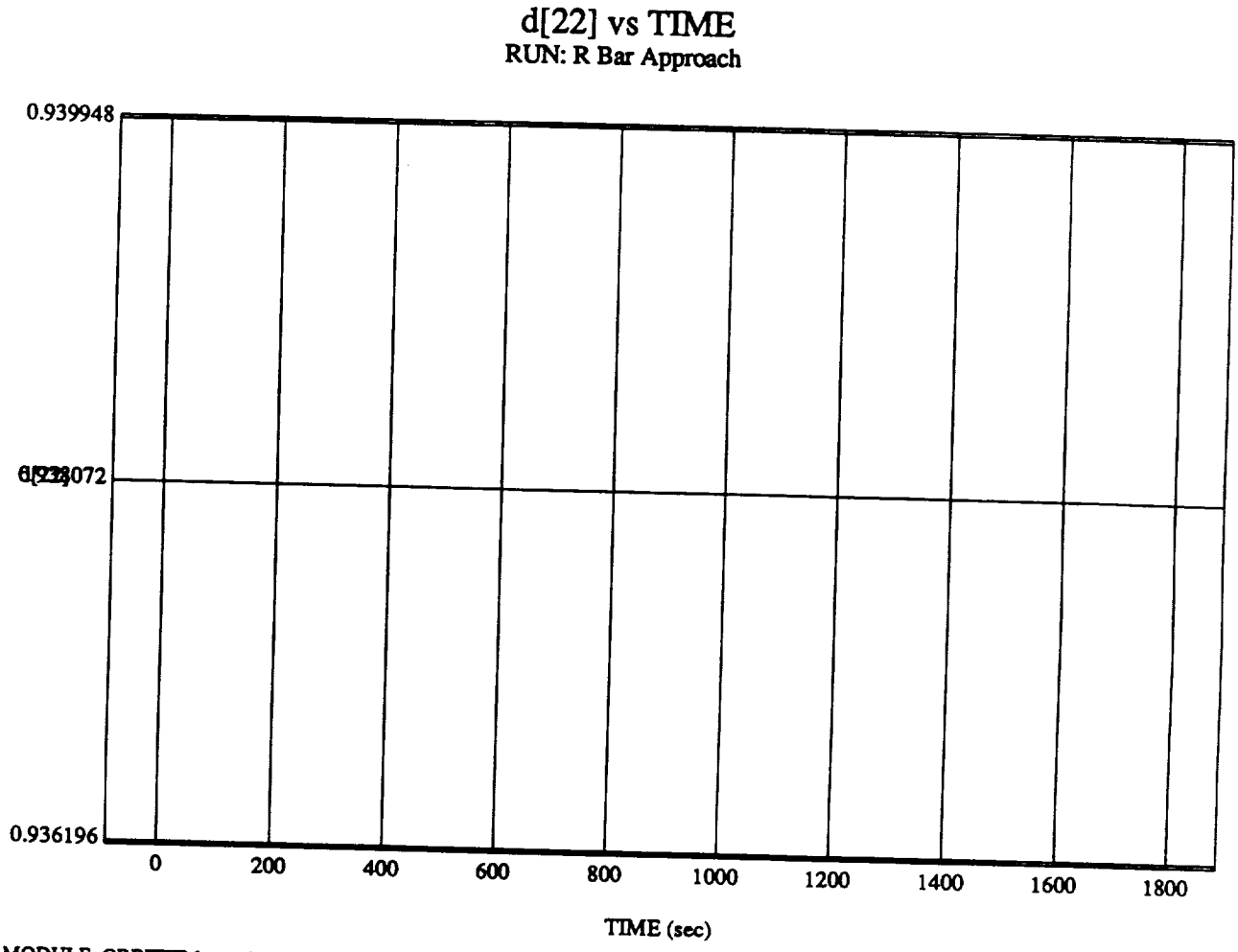
MODULE: ORBITER.lm_azim
DATA SAMPLING FREQUENCY: 0.200 Hz



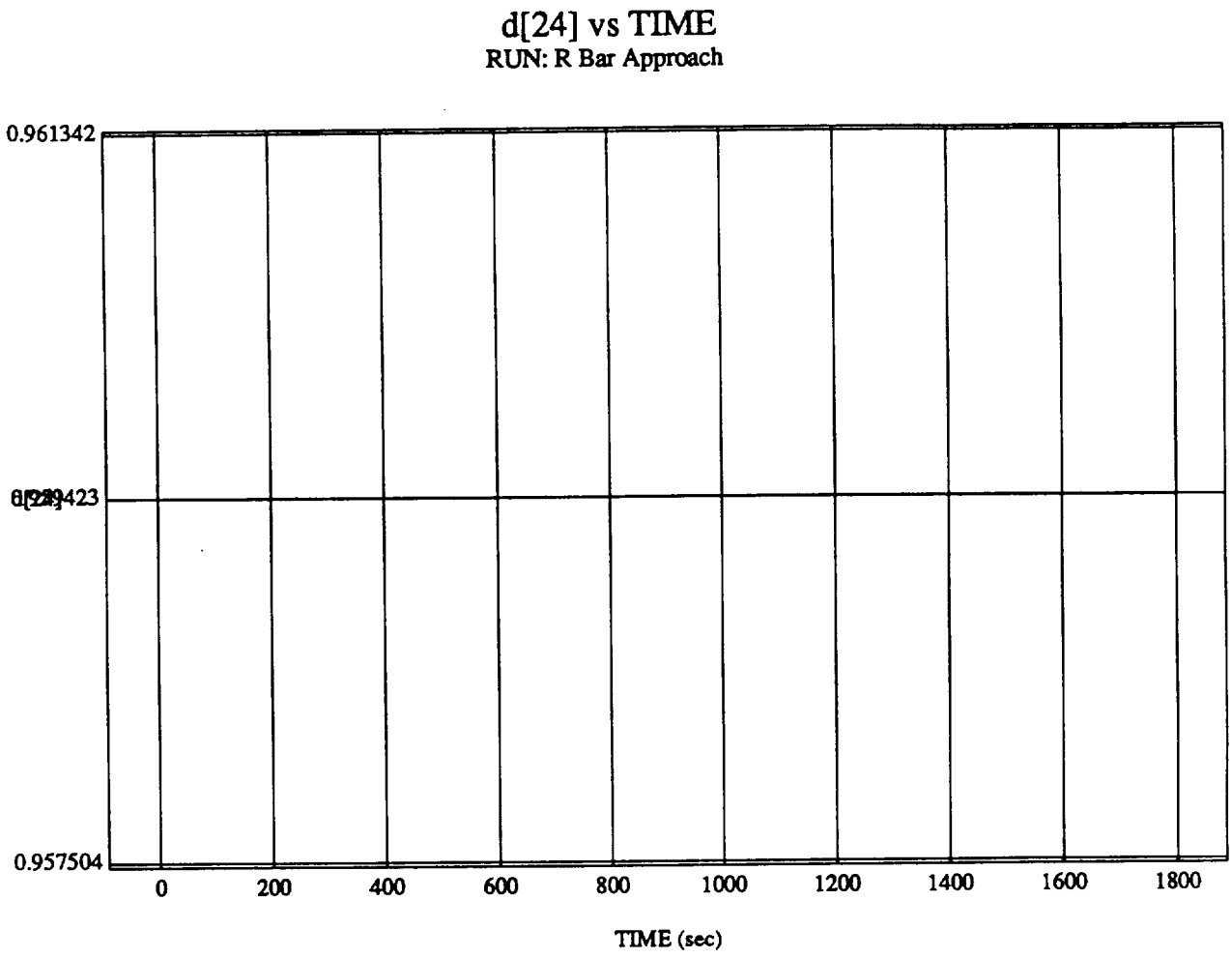
MODULE: ORBITER.lm_azim
DATA SAMPLING FREQUENCY: 0.200 Hz



MODULE: ORBITER.lm_azim
DATA SAMPLING FREQUENCY: 0.200 Hz

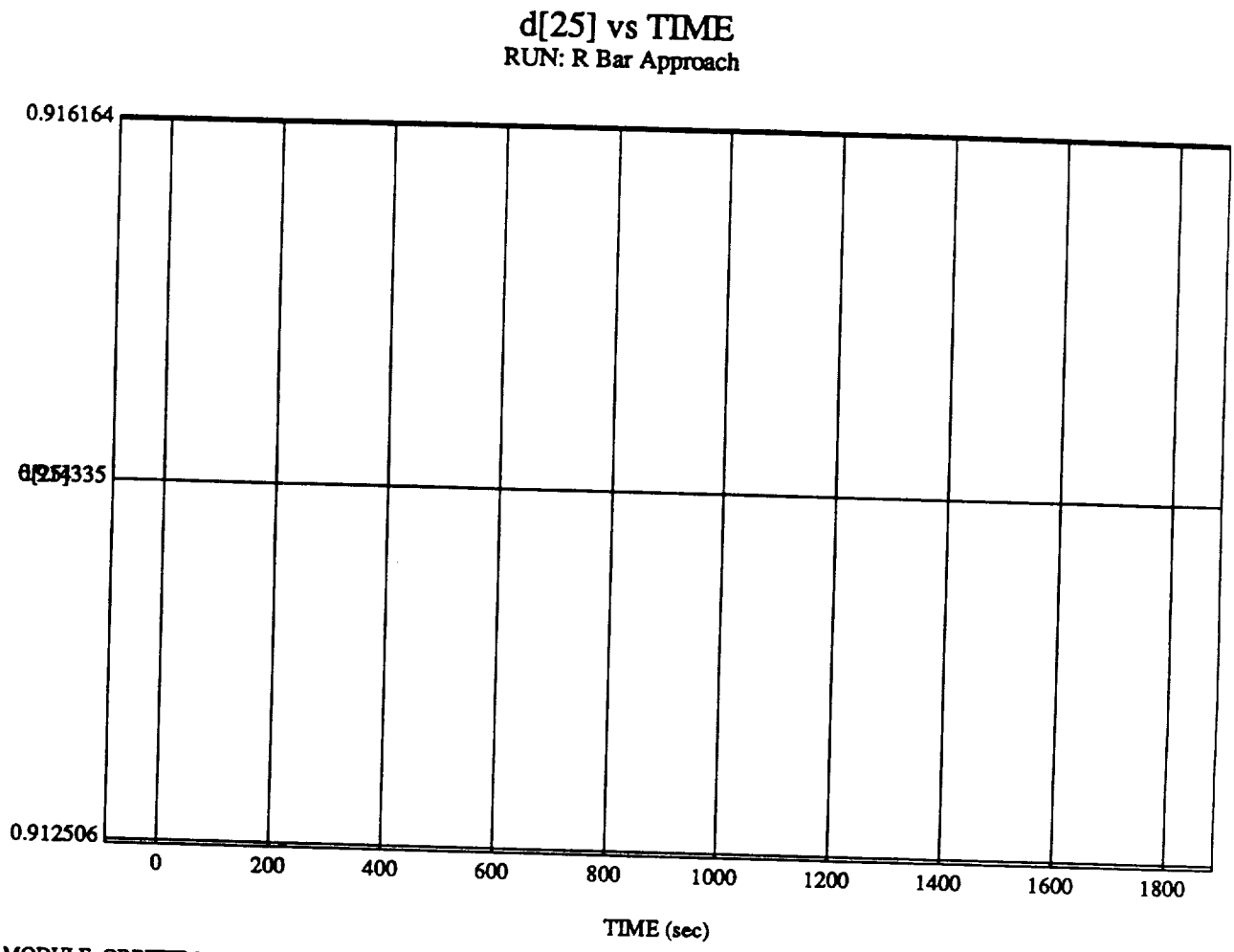


MODULE: ORBITER.lm_azim
DATA SAMPLING FREQUENCY: 0.200 Hz



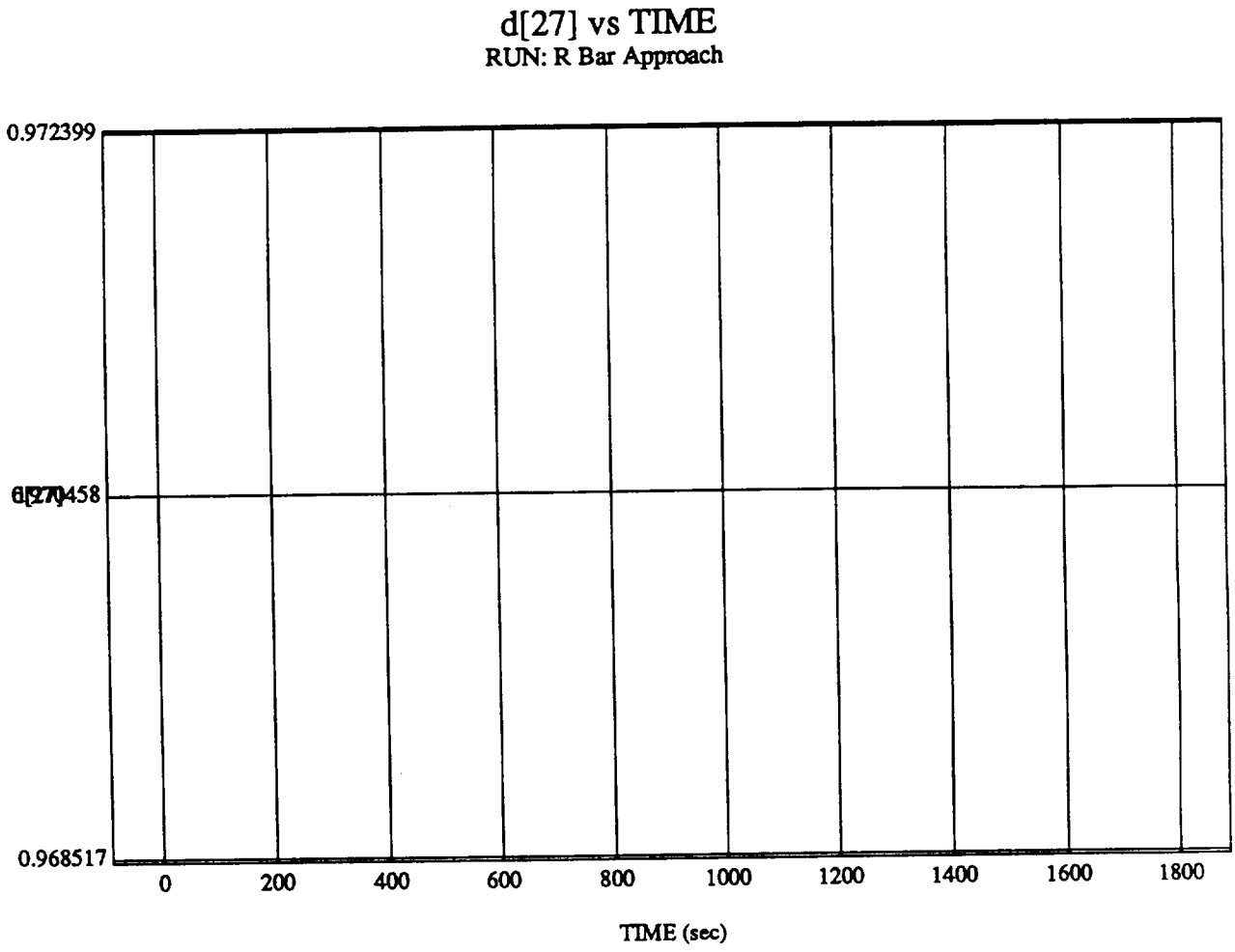
MODULE: ORBITER.lm_azim
DATA SAMPLING FREQUENCY: 0.200 Hz

SIMULATION APPLICATION: ARIC Translational Controller Simulation

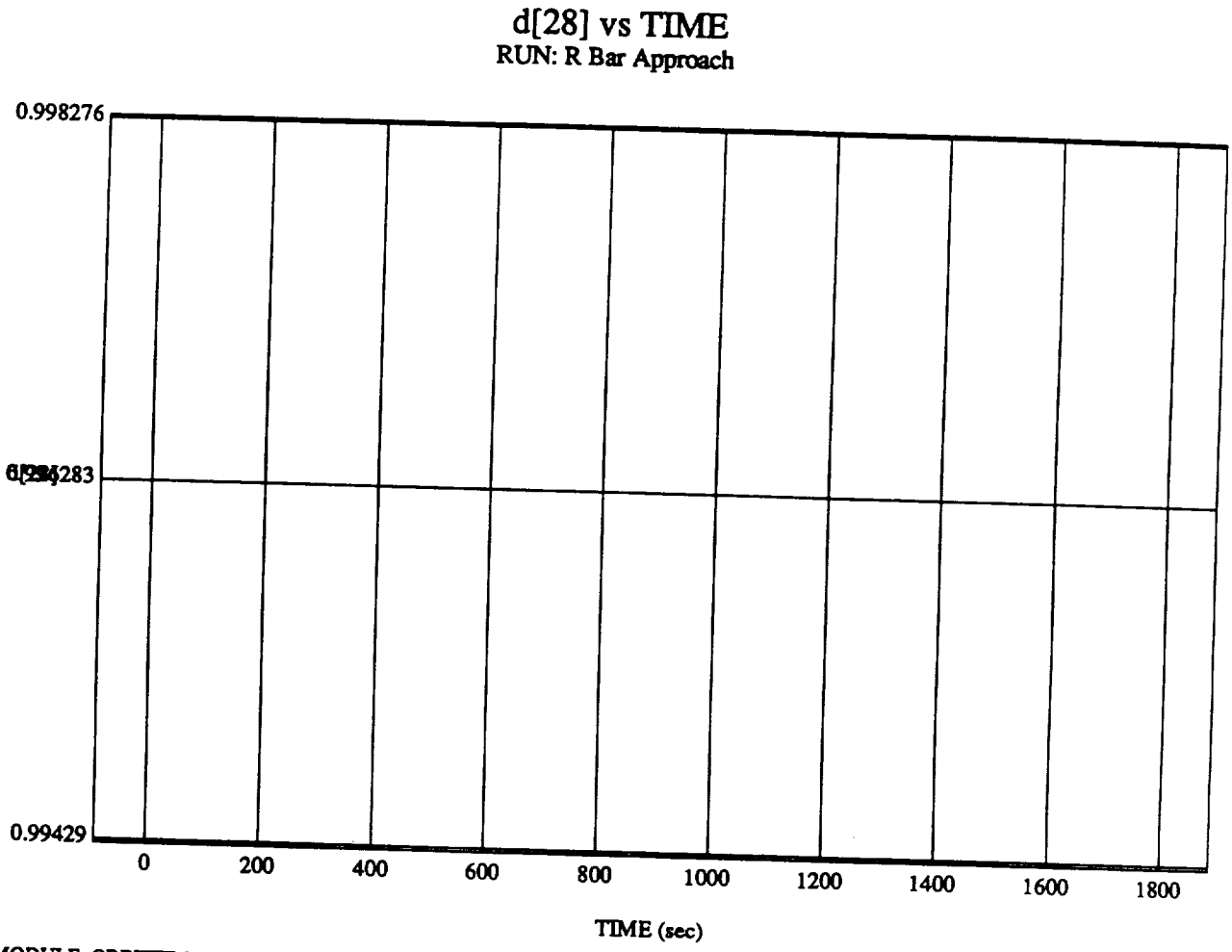


MODULE: ORBITER.lm_azim
DATA SAMPLING FREQUENCY: 0.200 Hz

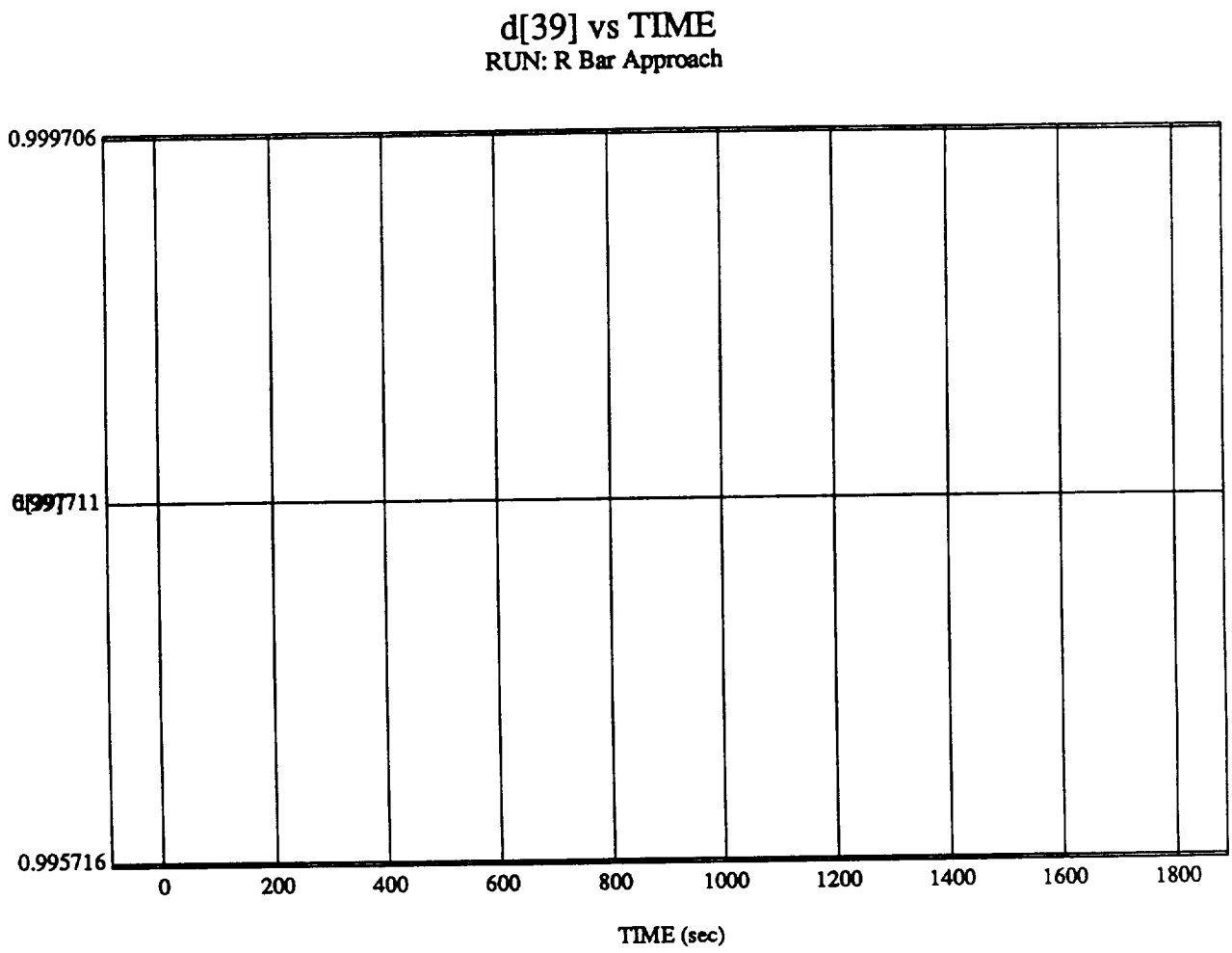
SIMULATION APPLICATION: ARIC Translational Controller Simulation



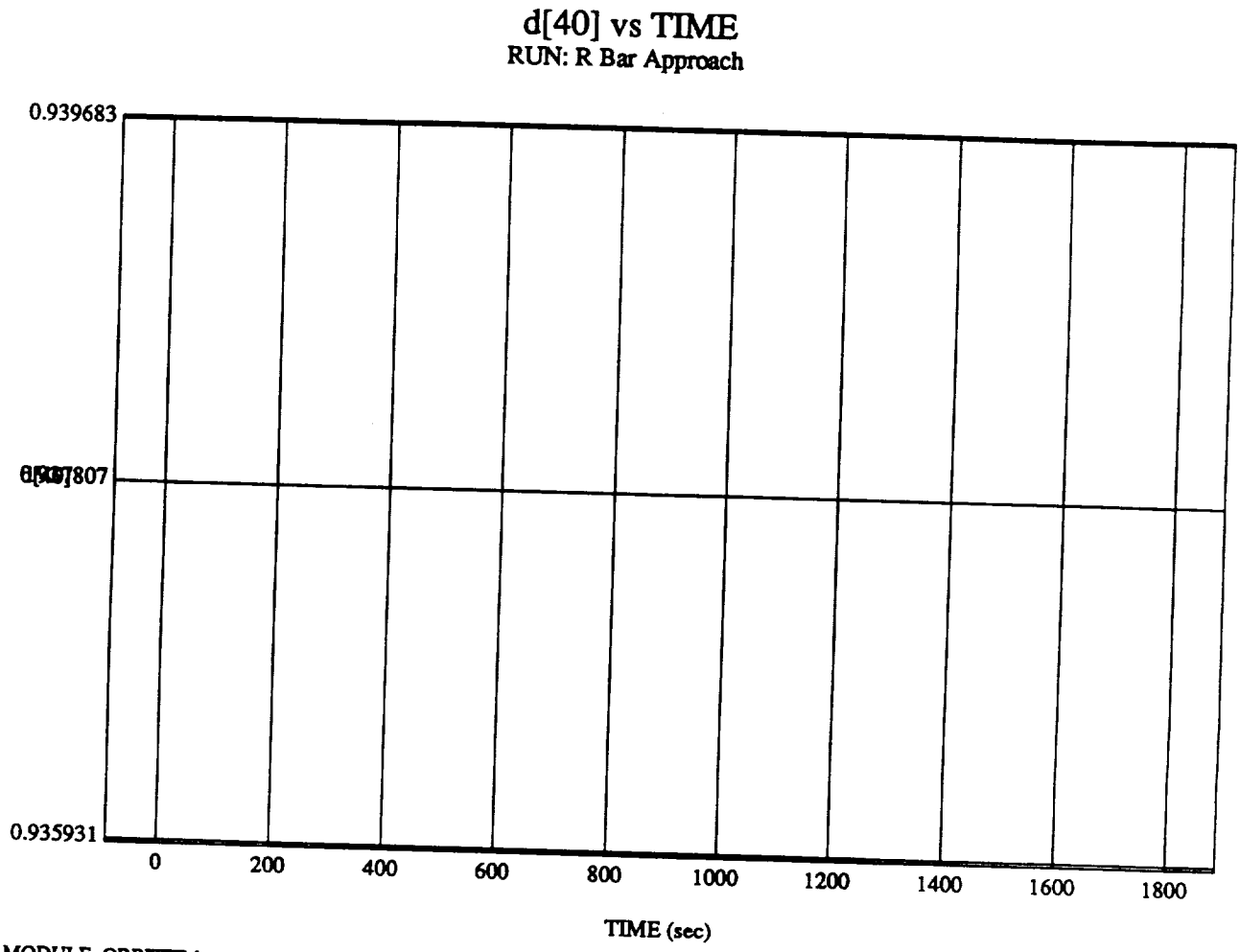
MODULE: ORBITER.lm_azim
DATA SAMPLING FREQUENCY: 0.200 Hz



MODULE: ORBITER.lm_azim
DATA SAMPLING FREQUENCY: 0.200 Hz



MODULE: ORBITER.lm_azim
DATA SAMPLING FREQUENCY: 0.200 Hz

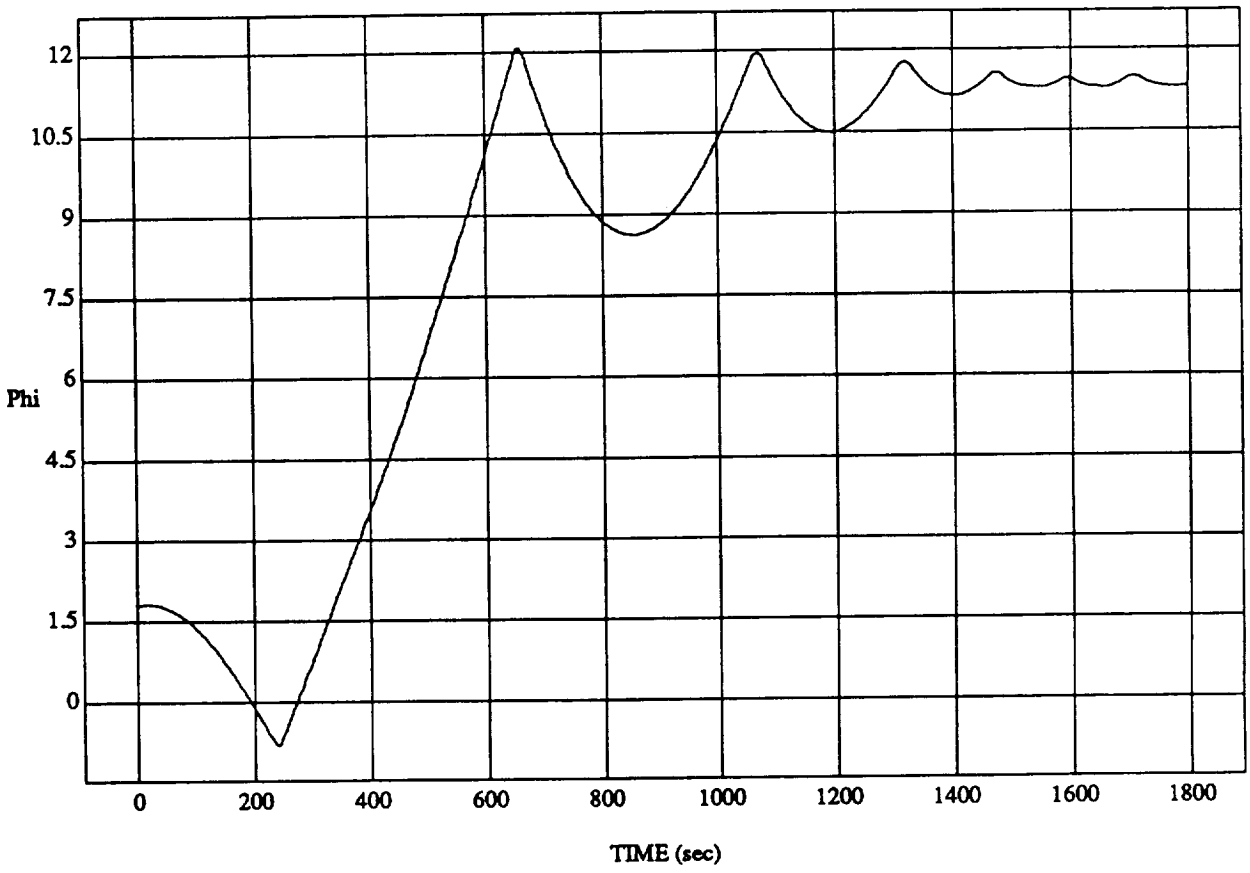


MODULE: ORBITER.lm_azim
DATA SAMPLING FREQUENCY: 0.200 Hz

SIMULATION APPLICATION: ARIC Translational Controller Simulation

Phi vs TIME

RUN: R Bar Approach



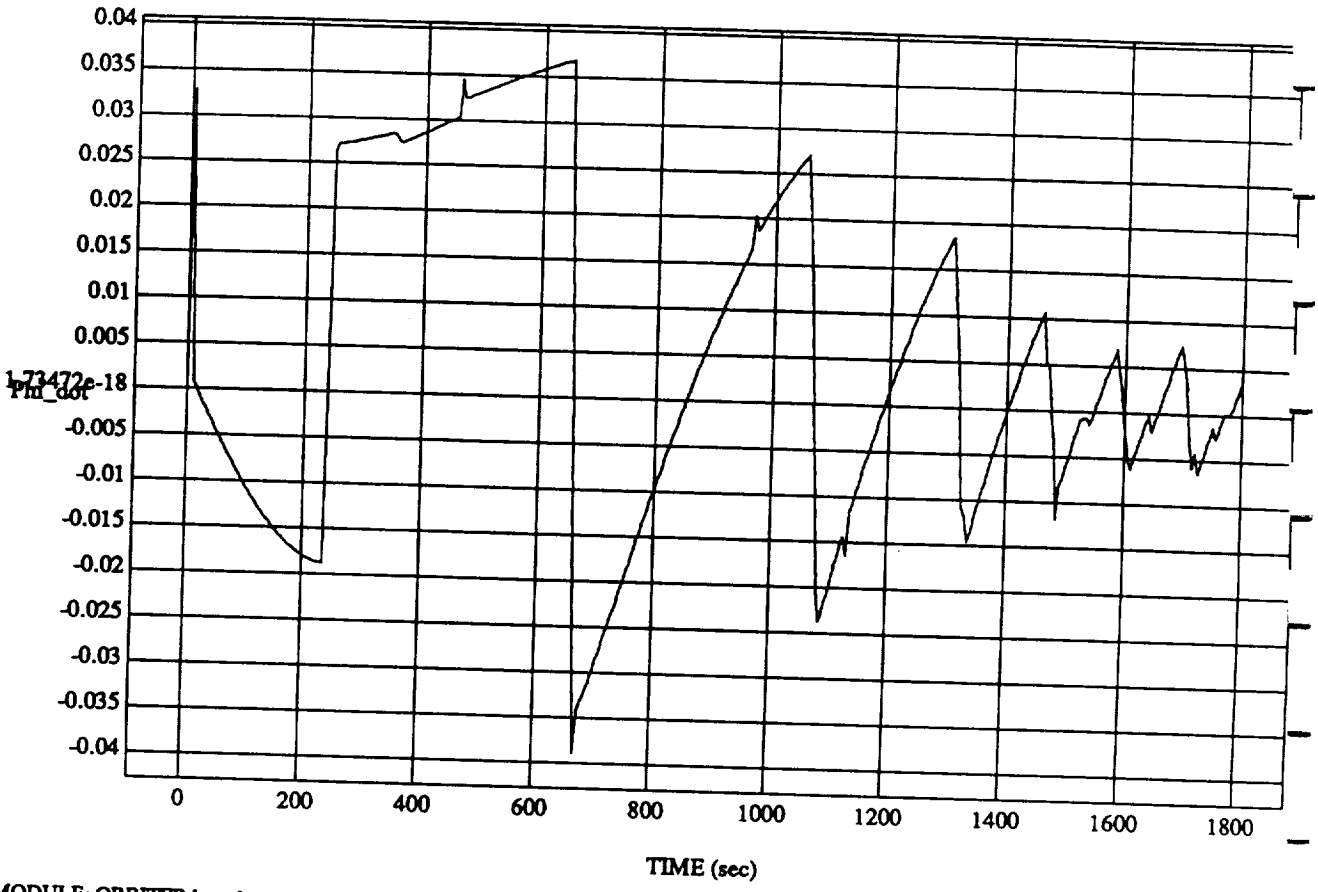
MODULE: ORBITER.lm_elev

DATA SAMPLING FREQUENCY: 0.200 Hz

SIMULATION APPLICATION: ARIC Translational Controller Simulation

Phi_dot vs TIME

RUN: R Bar Approach

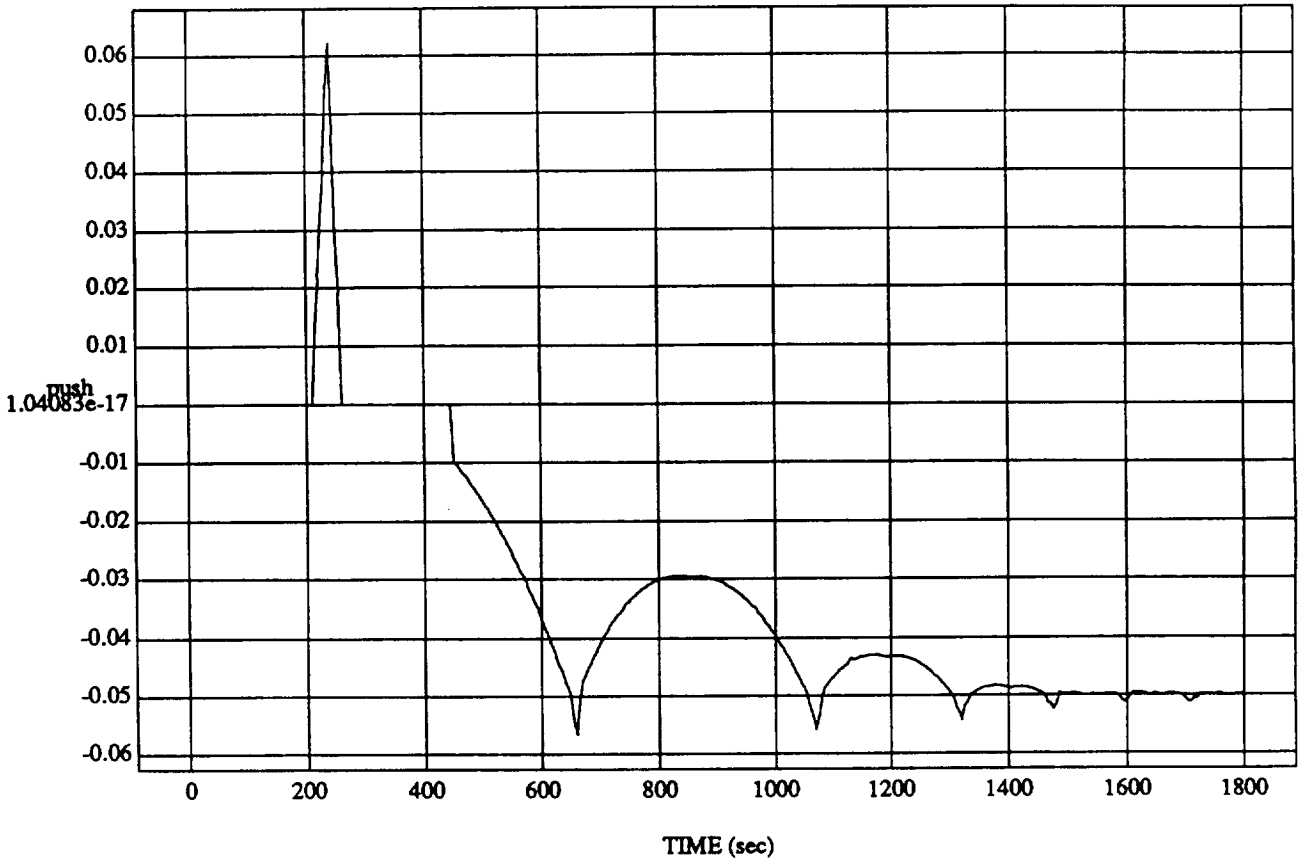


MODULE: ORBITER.lm_elev

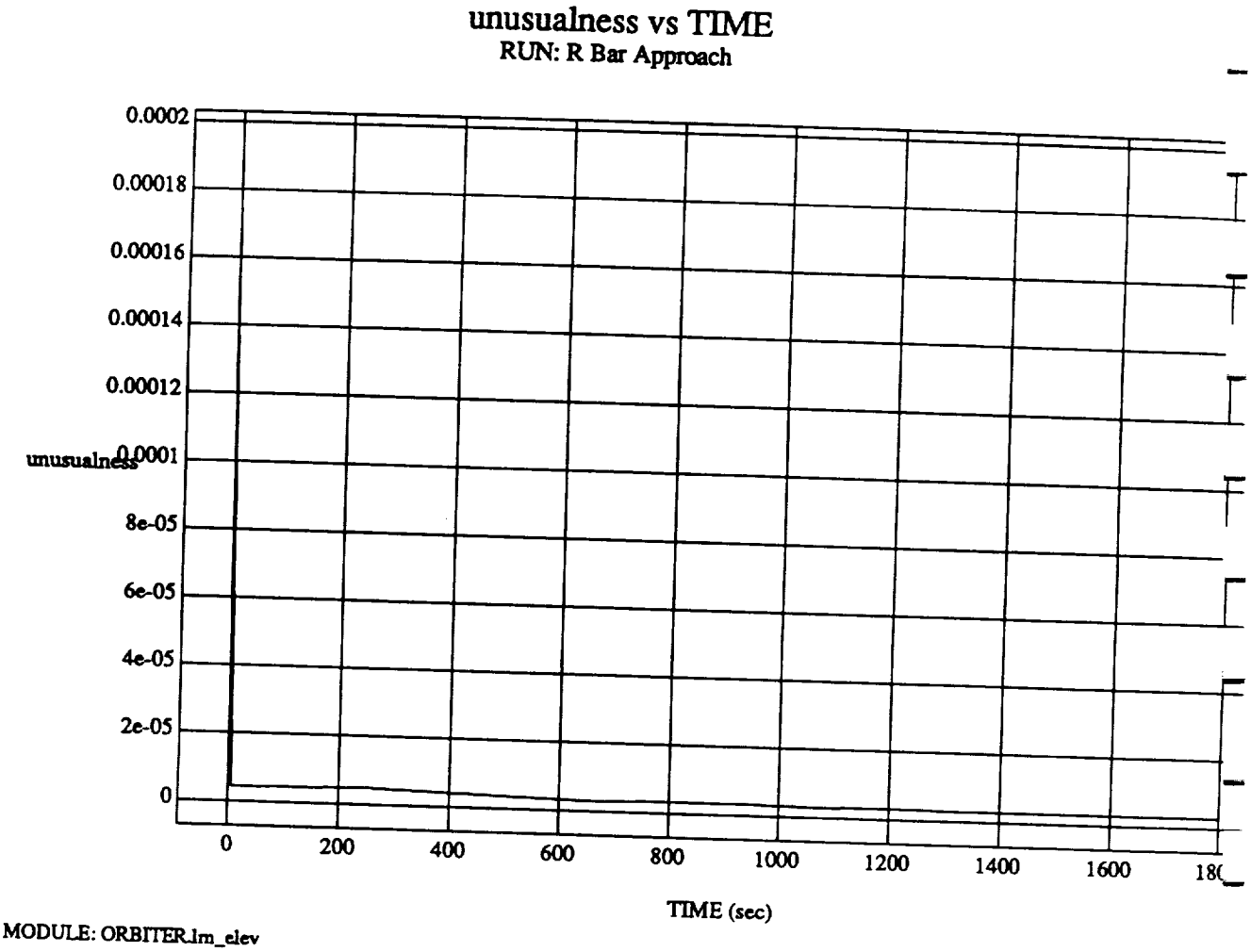
DATA SAMPLING FREQUENCY: 0.200 Hz

push vs TIME

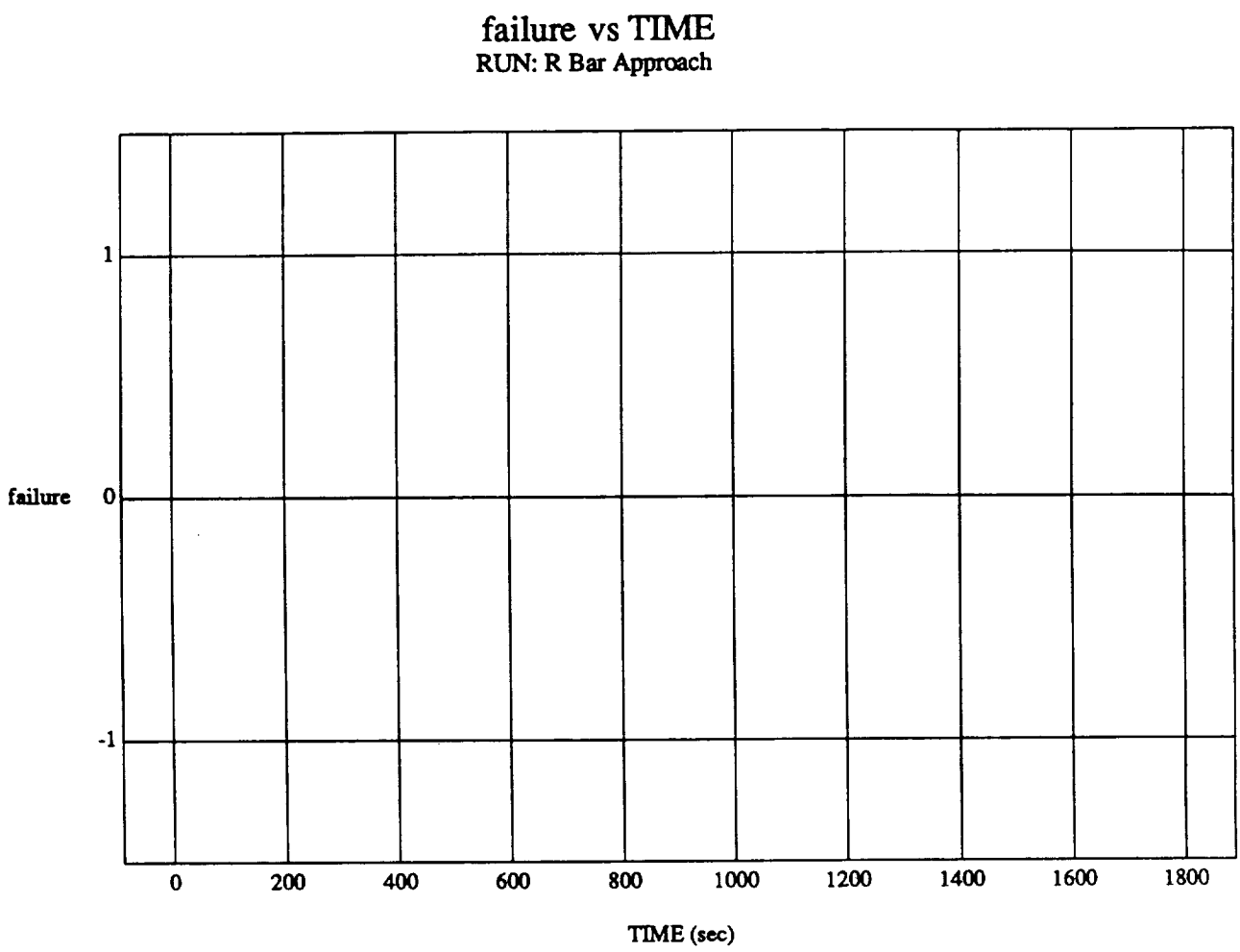
RUN: R Bar Approach



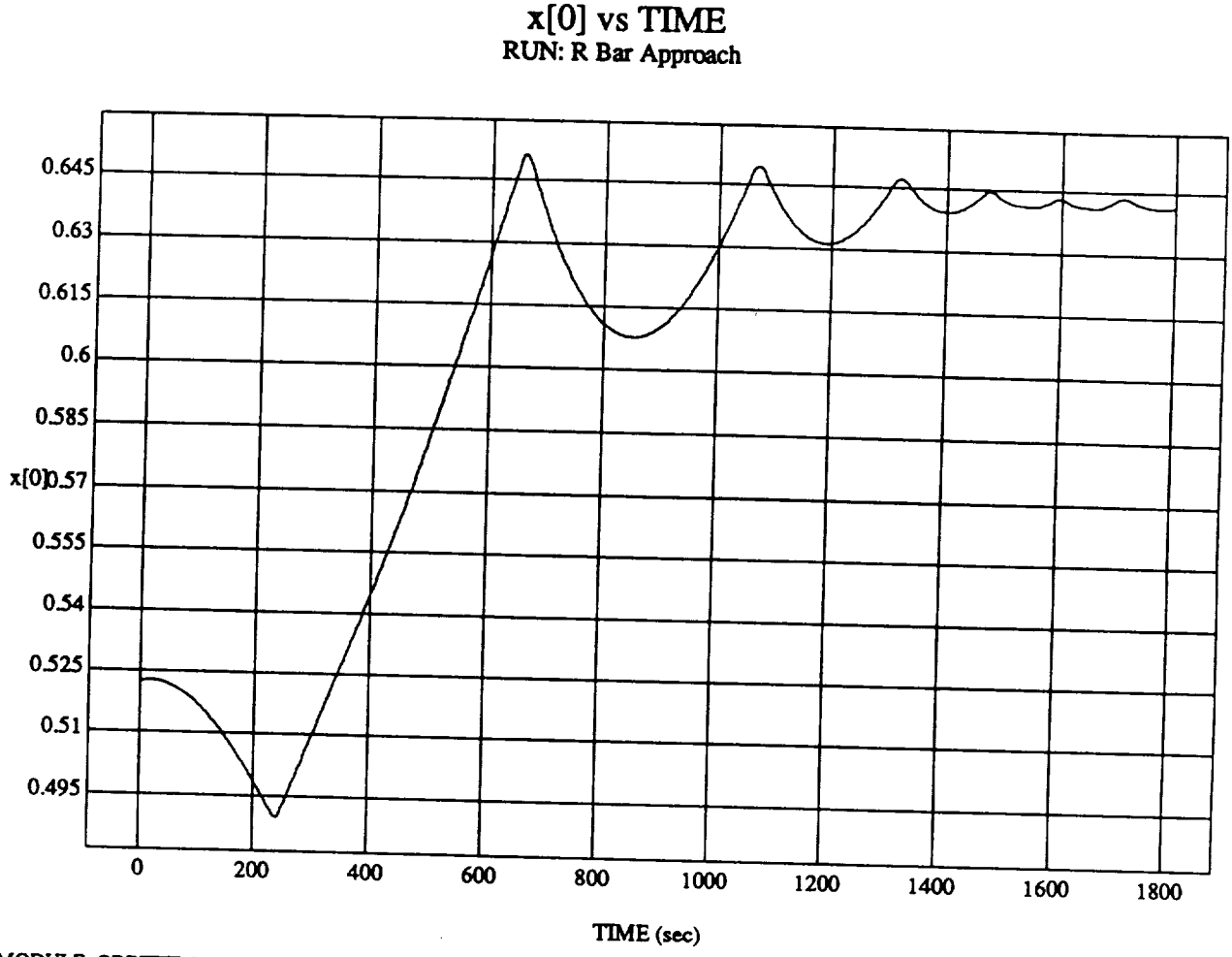
MODULE: ORBITER.lm_elev
DATA SAMPLING FREQUENCY: 0.200 Hz



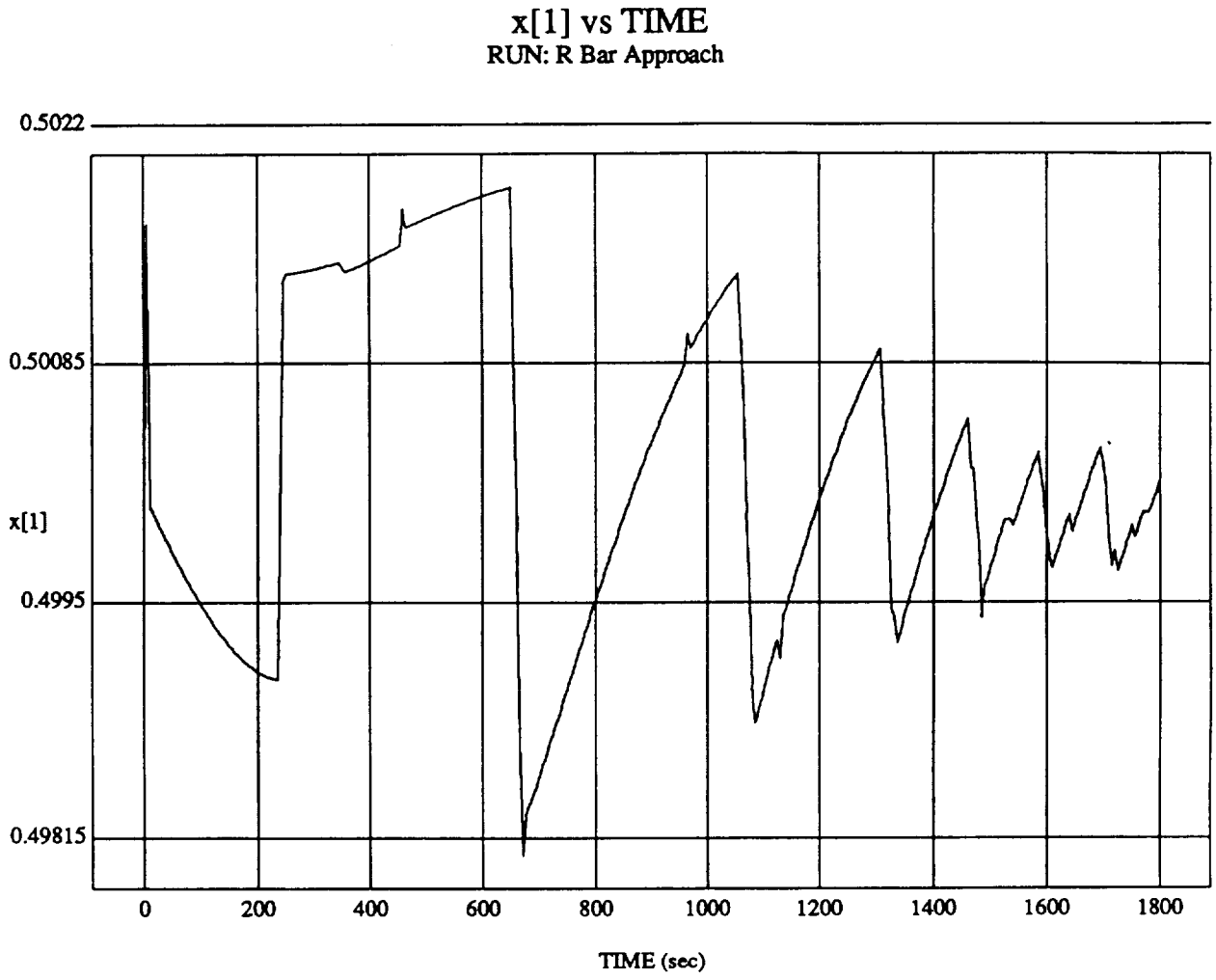
MODULE: ORBITER.lm_elev
DATA SAMPLING FREQUENCY: 0.200 Hz



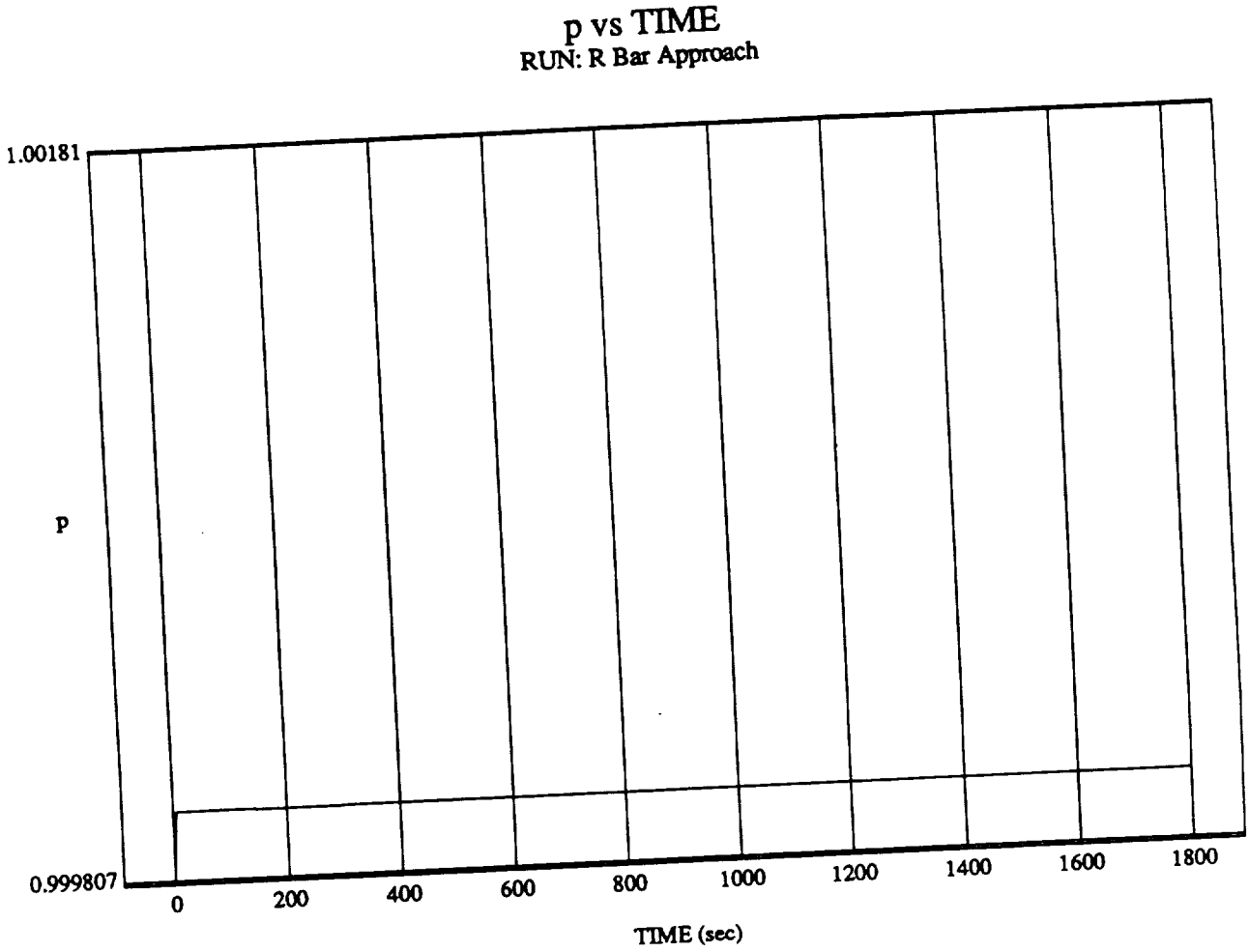
MODULE: ORBITER.lm_elev
DATA SAMPLING FREQUENCY: 0.200 Hz



MODULE: ORBITER.lm_elev
DATA SAMPLING FREQUENCY: 0.200 Hz



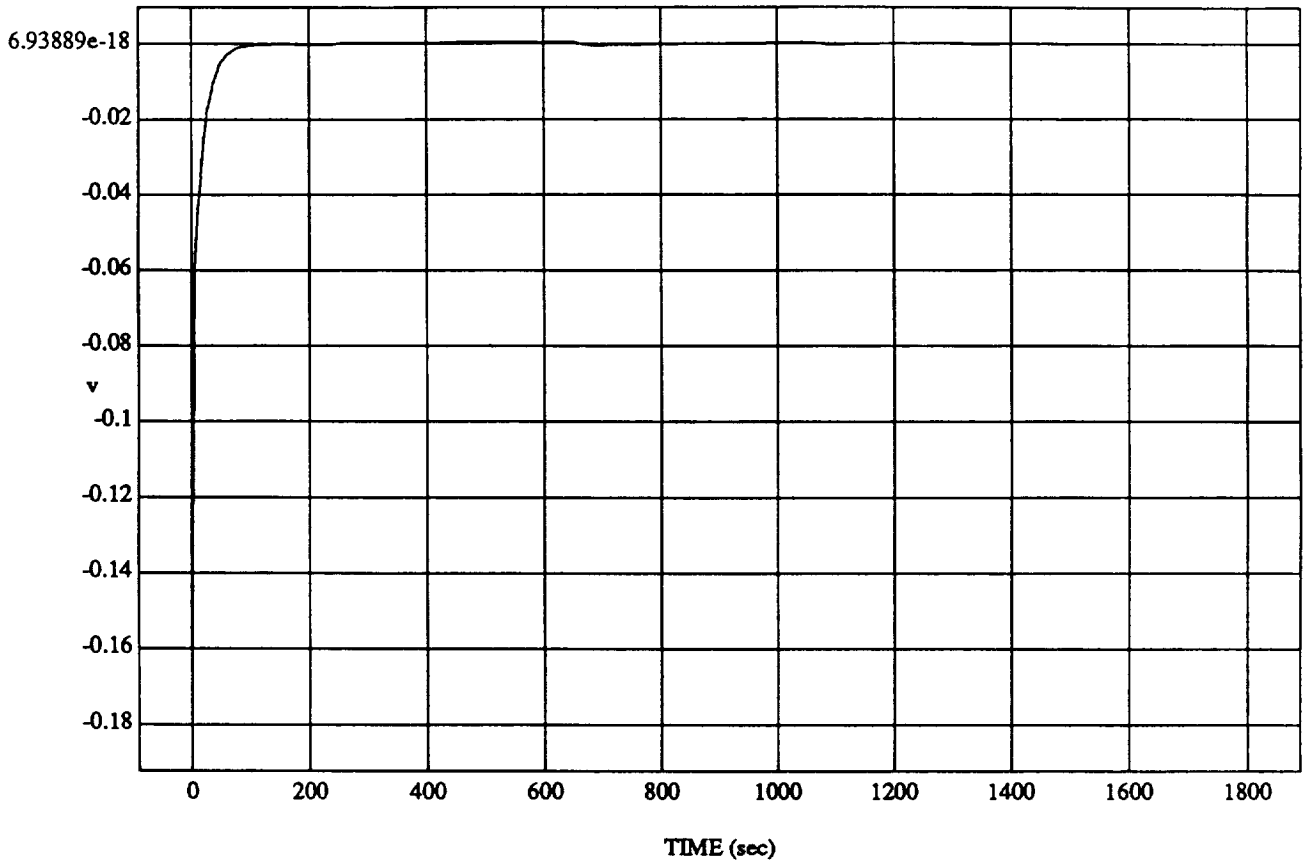
MODULE: ORBITER.lm_elev
DATA SAMPLING FREQUENCY: 0.200 Hz



MODULE: ORBITER.lm_elev
DATA SAMPLING FREQUENCY: 0.200 Hz

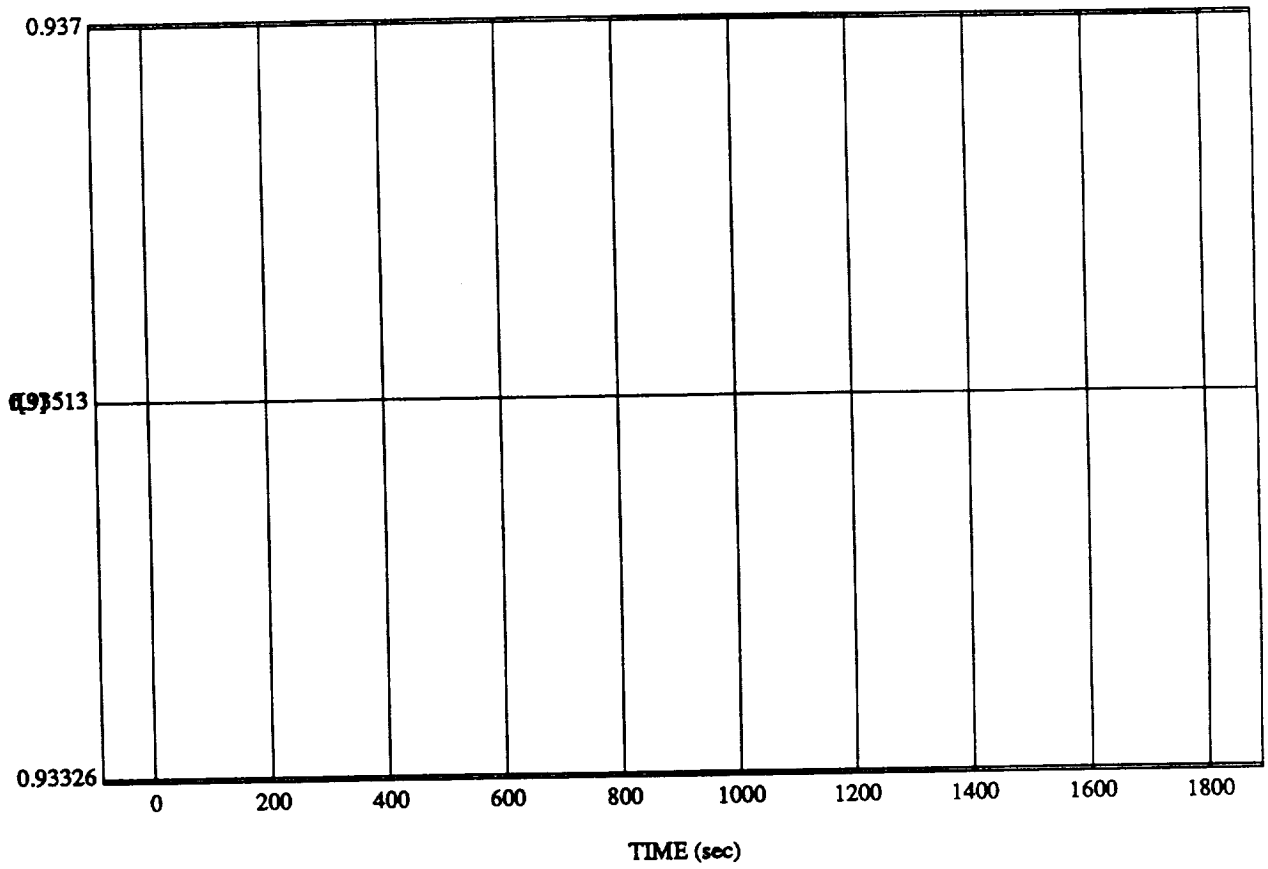
v vs TIME

RUN: R Bar Approach



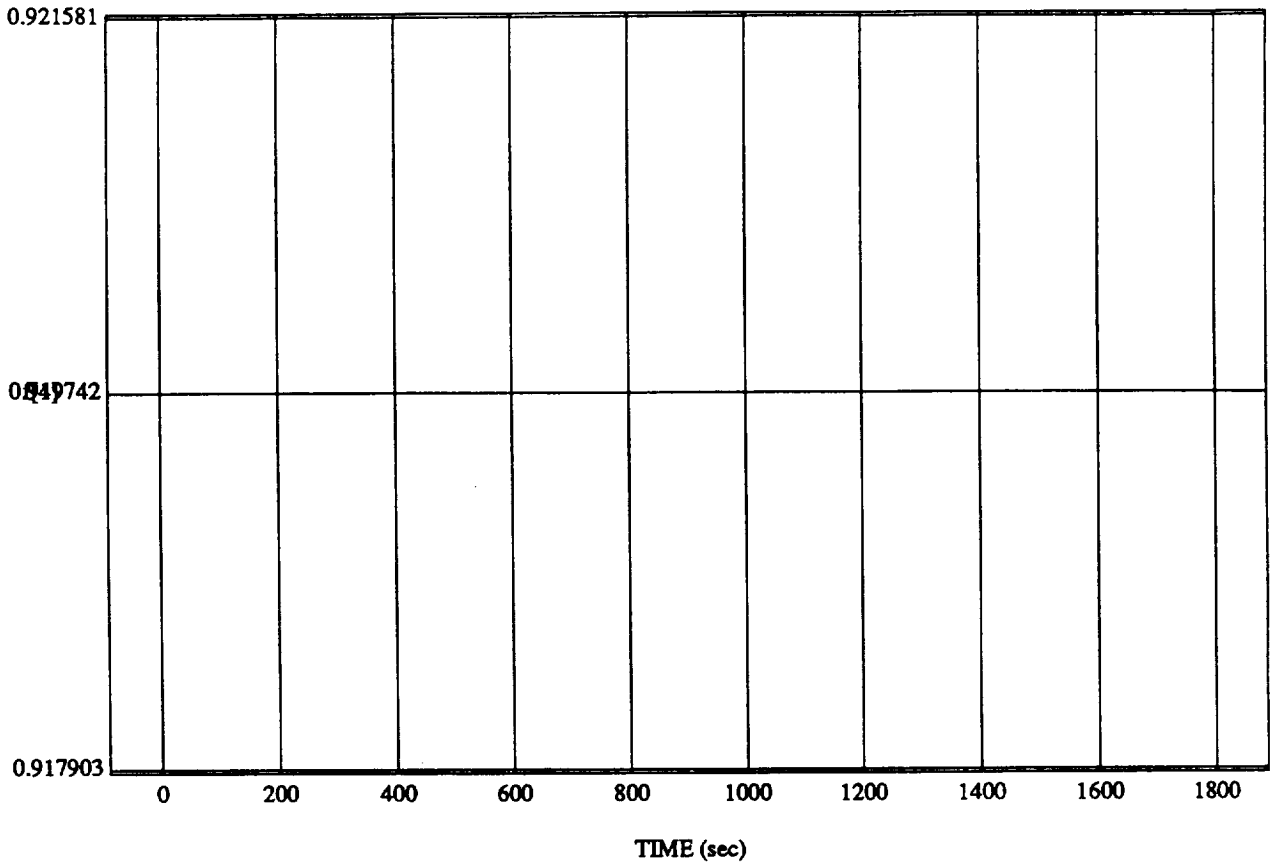
MODULE: ORBITER.lm_elev
DATA SAMPLING FREQUENCY: 0.200 Hz

f[3] vs TIME
RUN: R Bar Approach

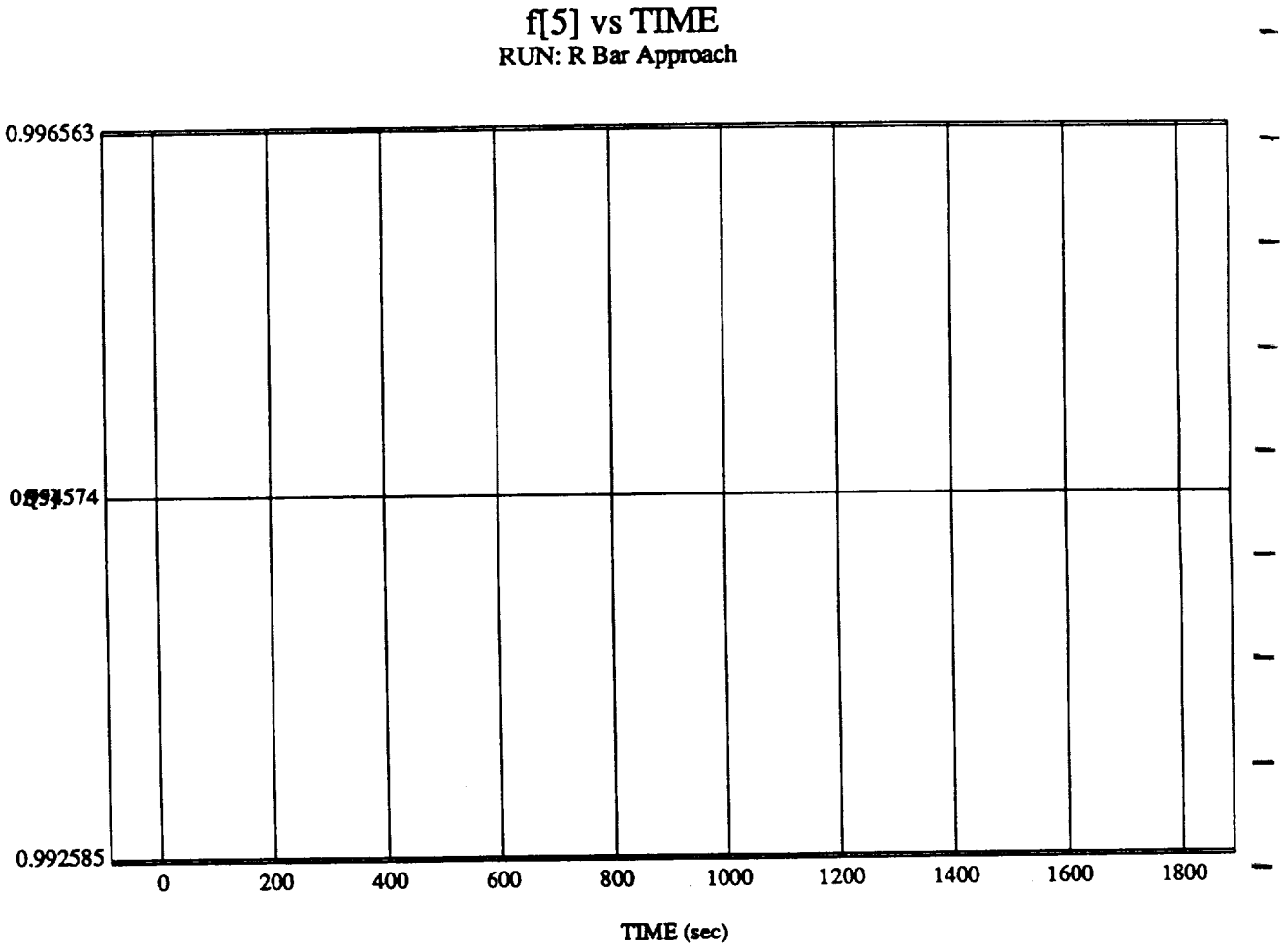


MODULE: ORBITER.lm_elev
DATA SAMPLING FREQUENCY: 0.200 Hz

f[4] vs TIME
RUN: R Bar Approach

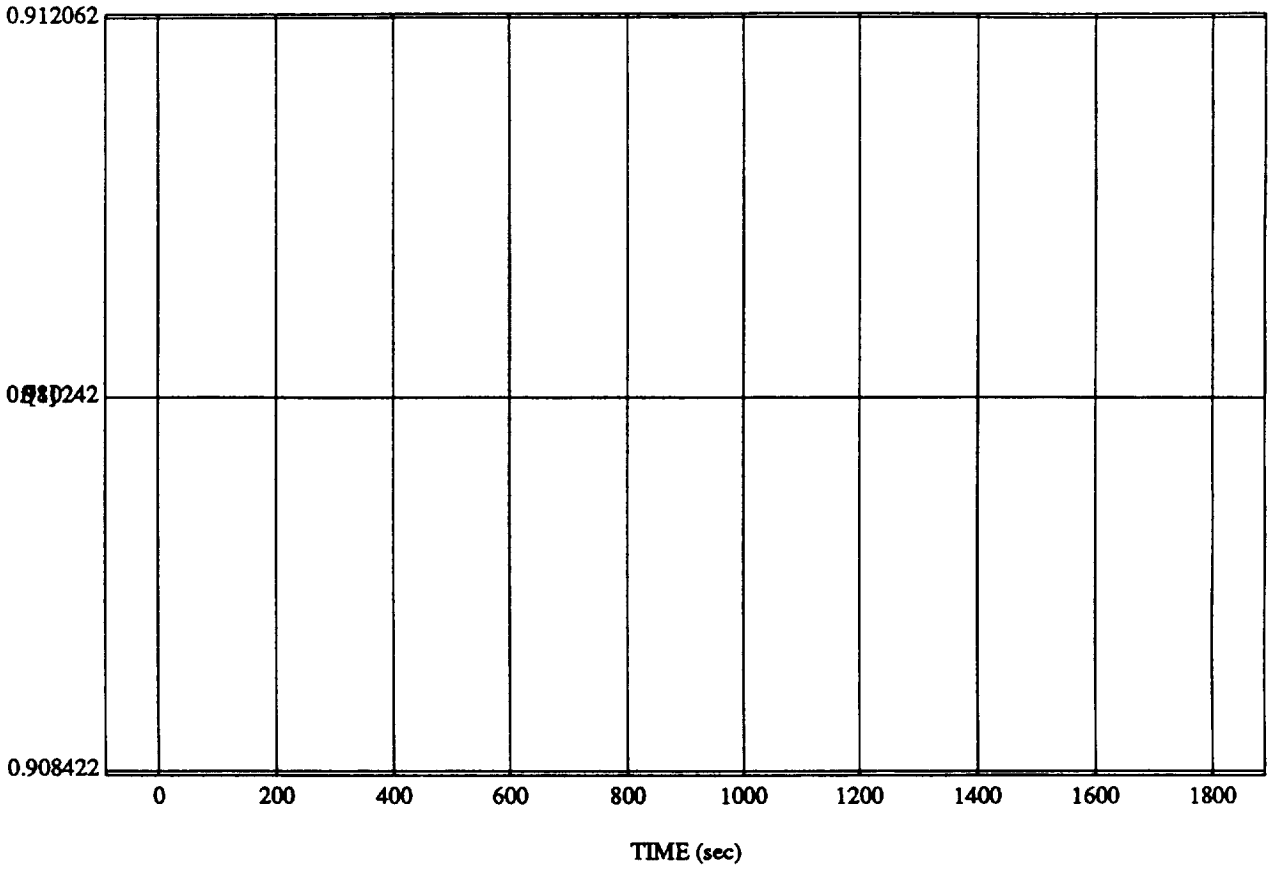


MODULE: ORBITER.lm_elev
DATA SAMPLING FREQUENCY: 0.200 Hz

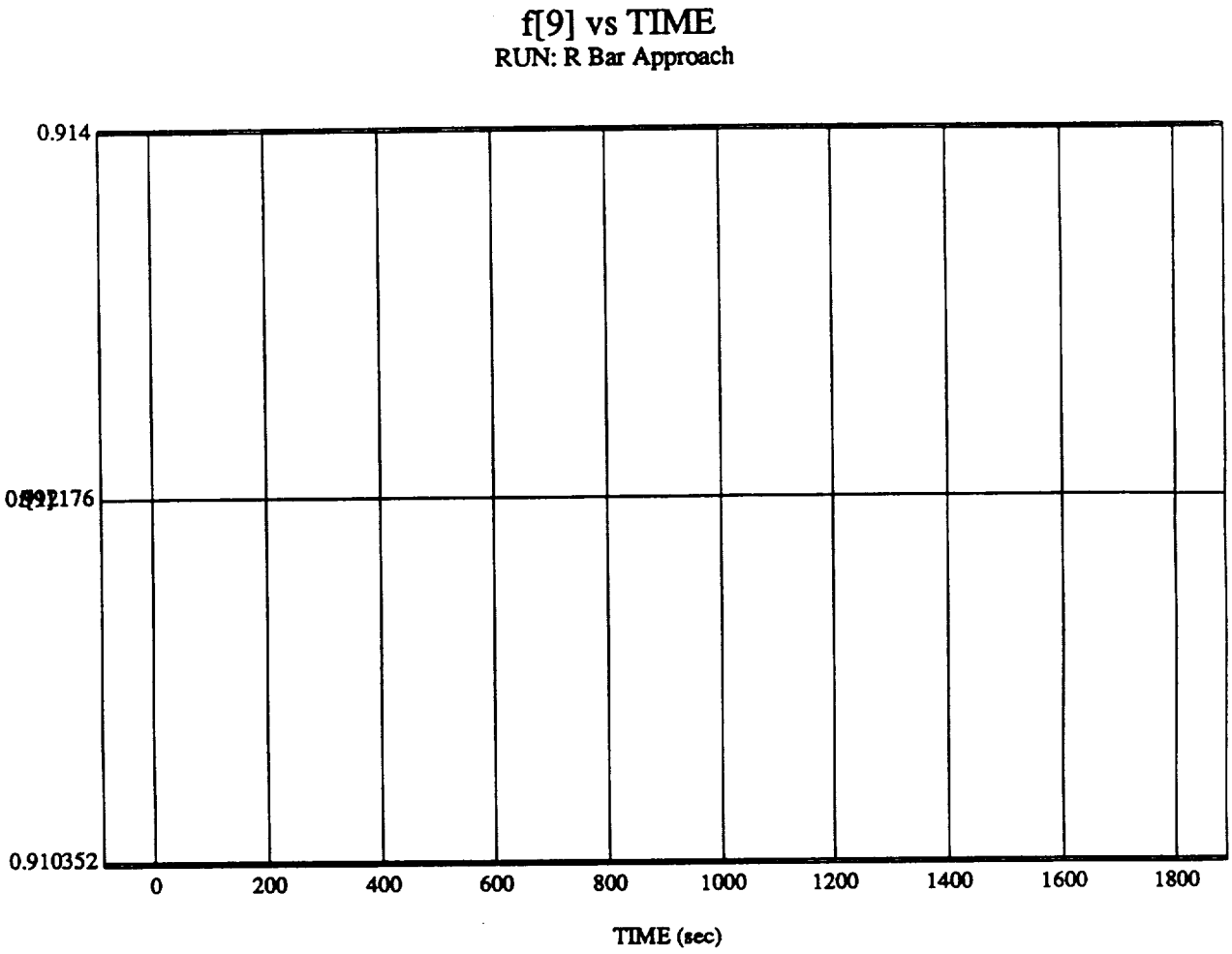


MODULE: ORBITER.lm_elev
DATA SAMPLING FREQUENCY: 0.200 Hz

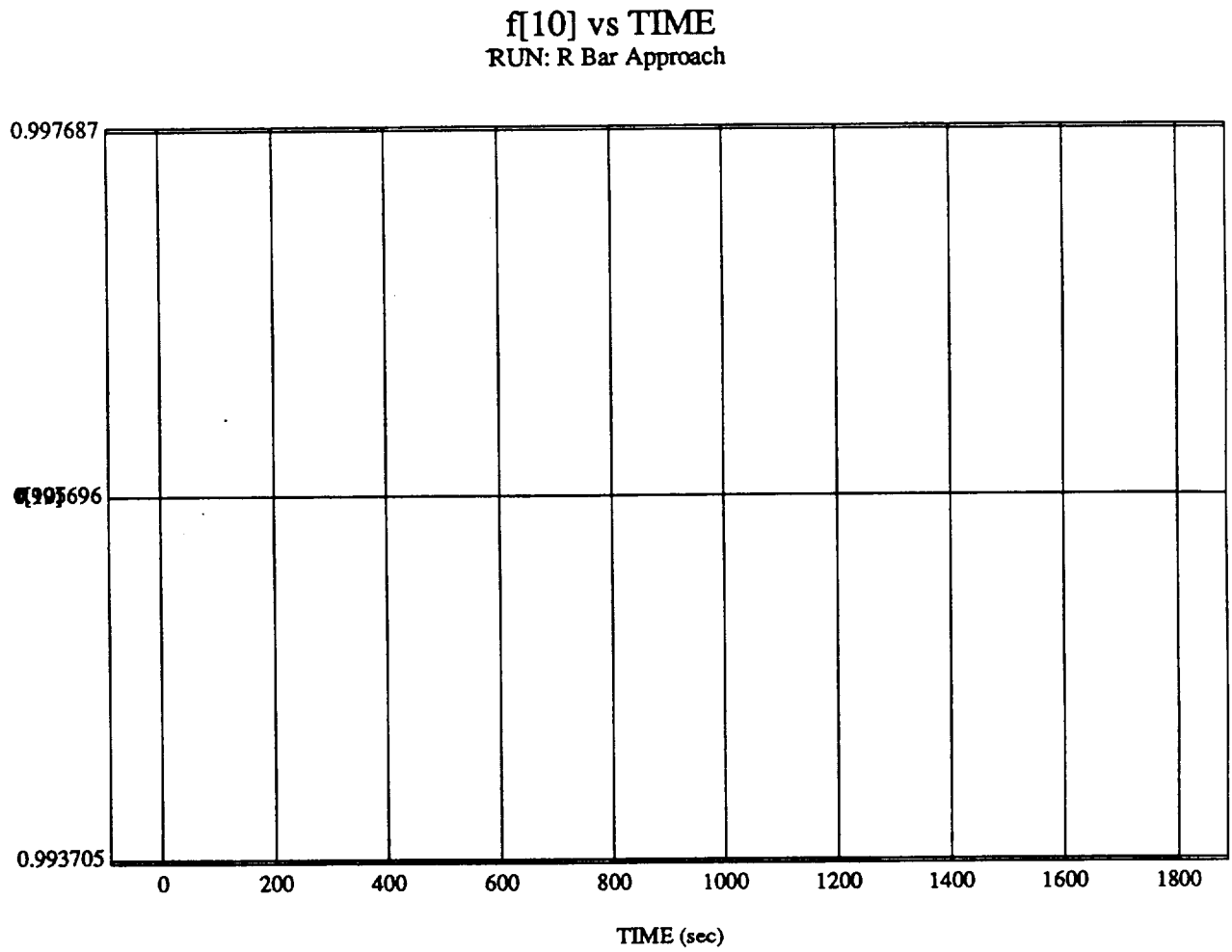
f[8] vs TIME
RUN: R Bar Approach



MODULE: ORBITER.lm_elev
DATA SAMPLING FREQUENCY: 0.200 Hz

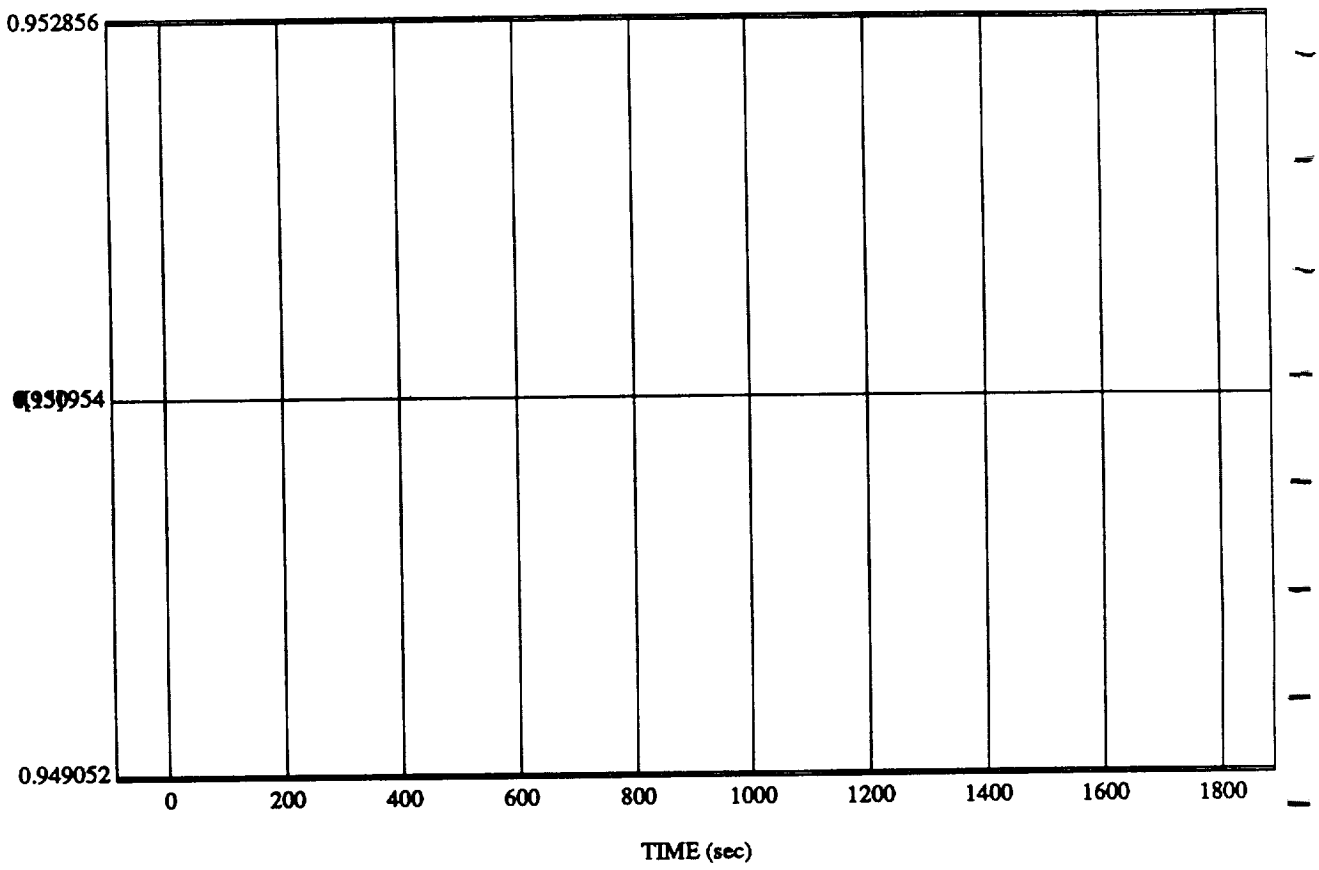


MODULE: ORBITER.lm_elev
DATA SAMPLING FREQUENCY: 0.200 Hz



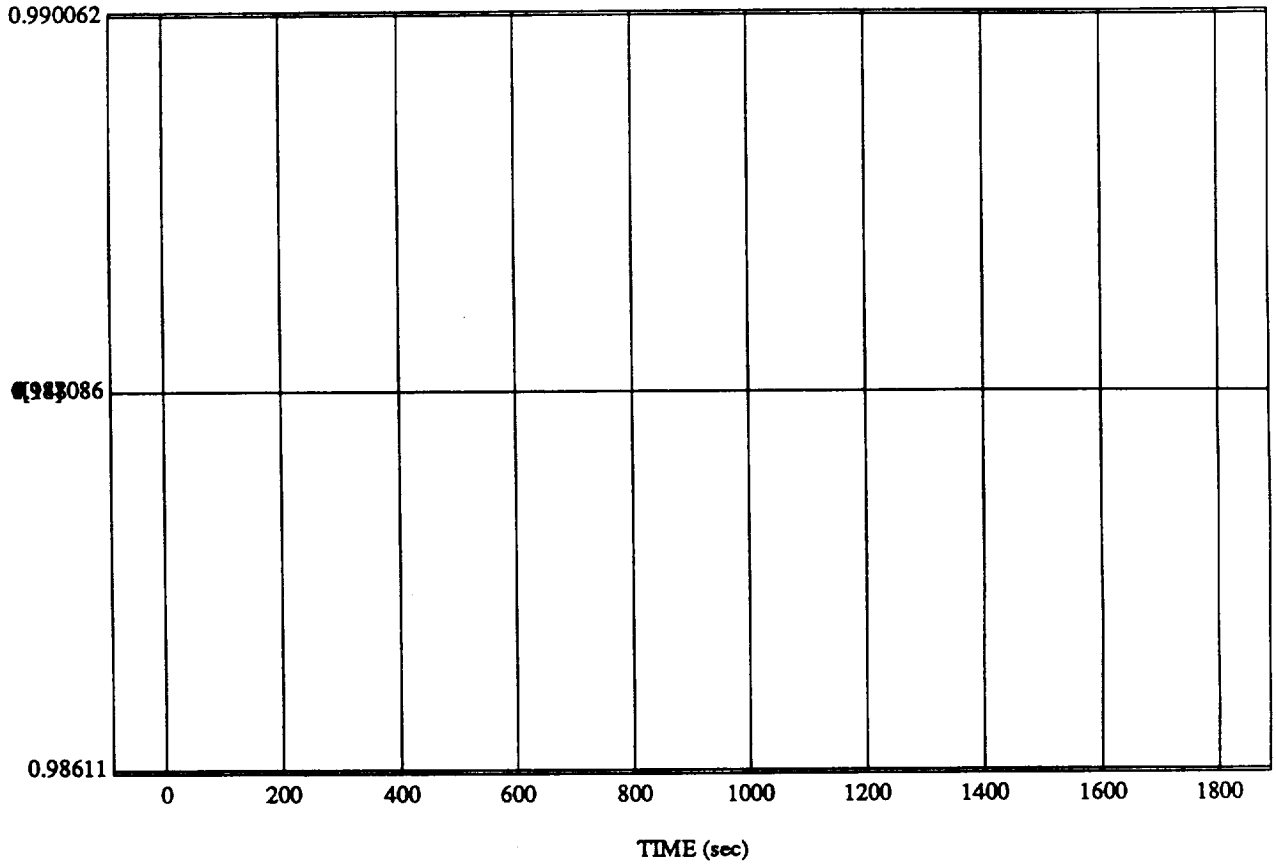
MODULE: ORBITER.lm_elev
DATA SAMPLING FREQUENCY: 0.200 Hz

f[13] vs TIME
RUN: R Bar Approach



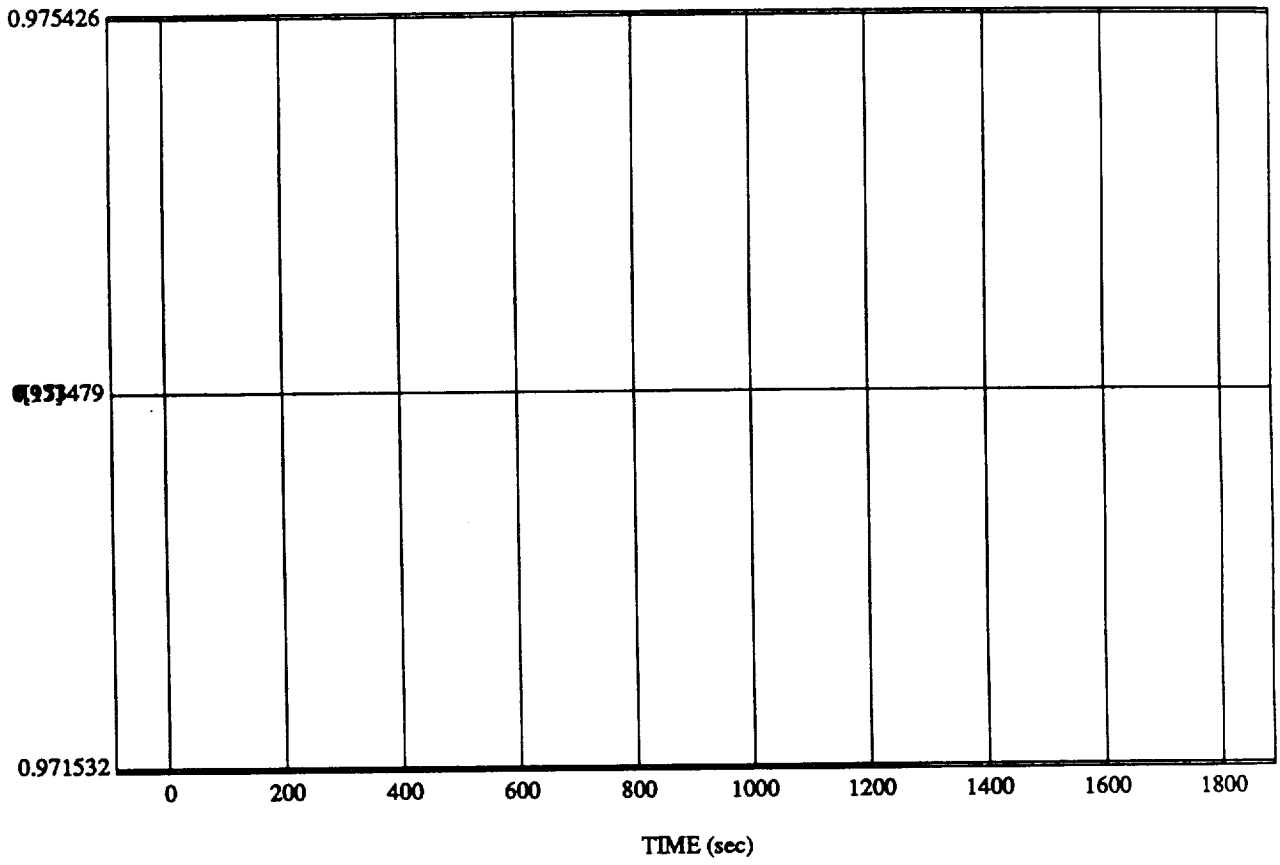
MODULE: ORBITER.lm_elev
DATA SAMPLING FREQUENCY: 0.200 Hz

f[14] vs TIME
RUN: R Bar Approach



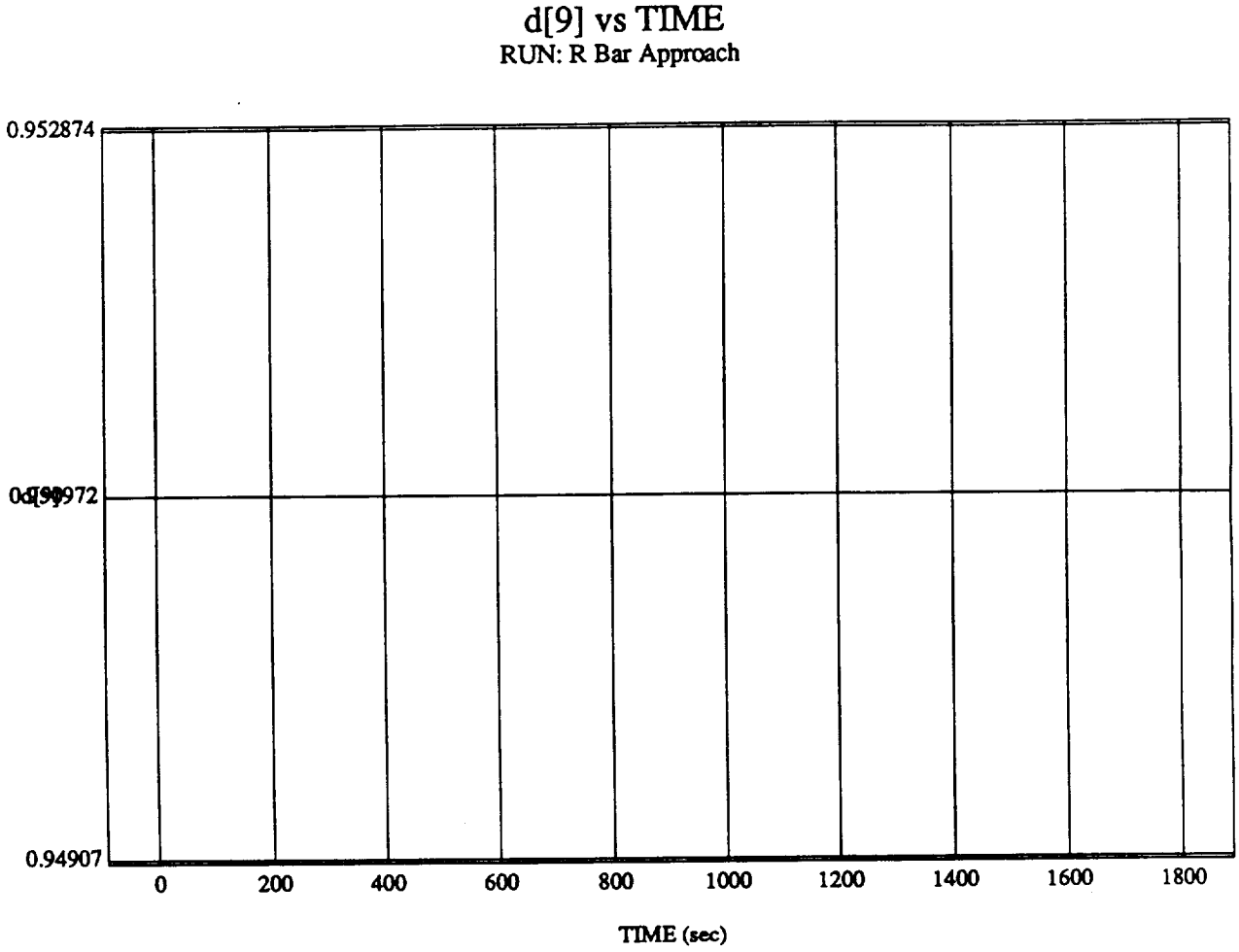
MODULE: ORBITER.lm_elev
DATA SAMPLING FREQUENCY: 0.200 Hz

f[15] vs TIME
RUN: R Bar Approach

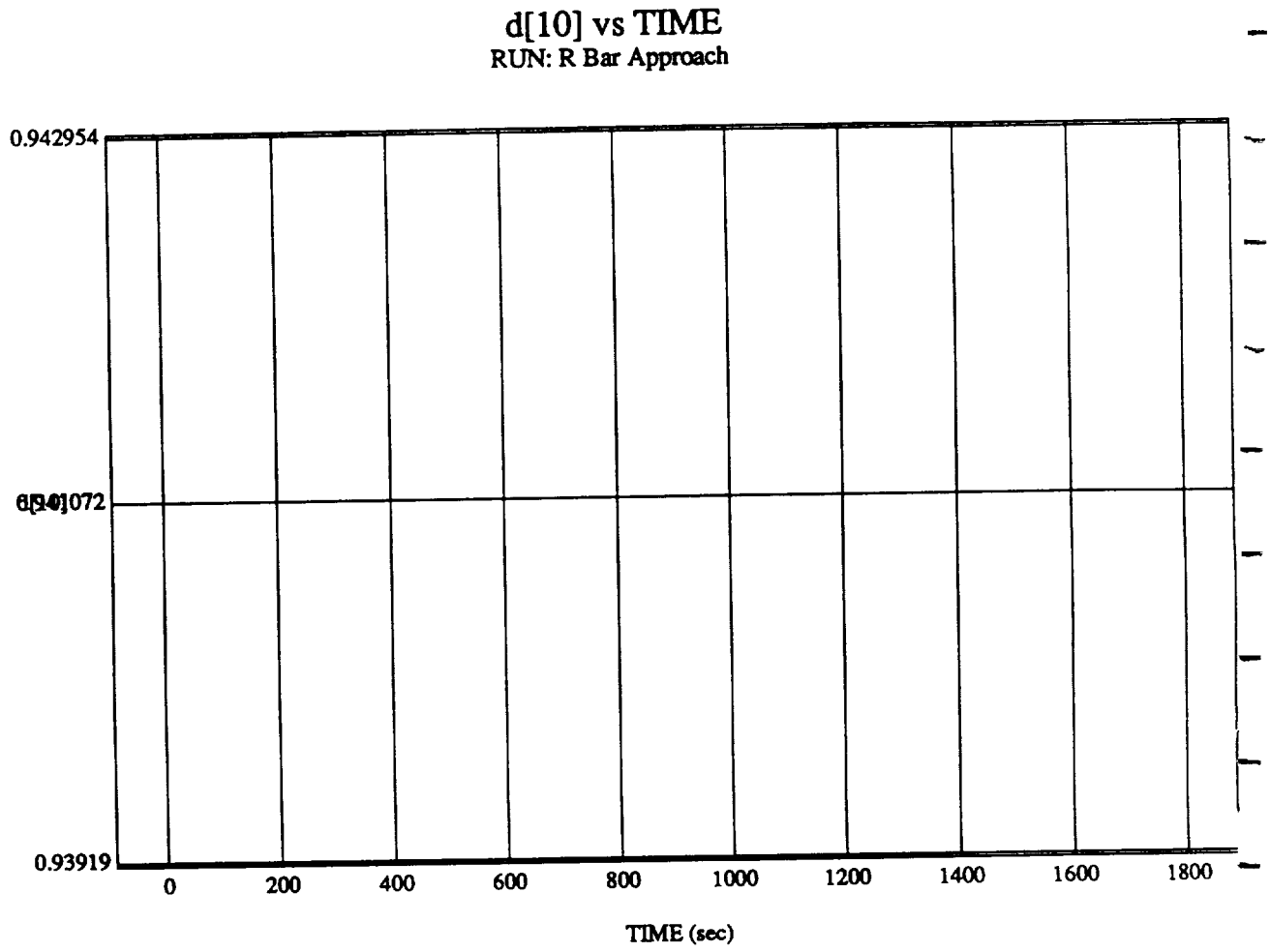


MODULE: ORBITER.lm_elev
DATA SAMPLING FREQUENCY: 0.200 Hz

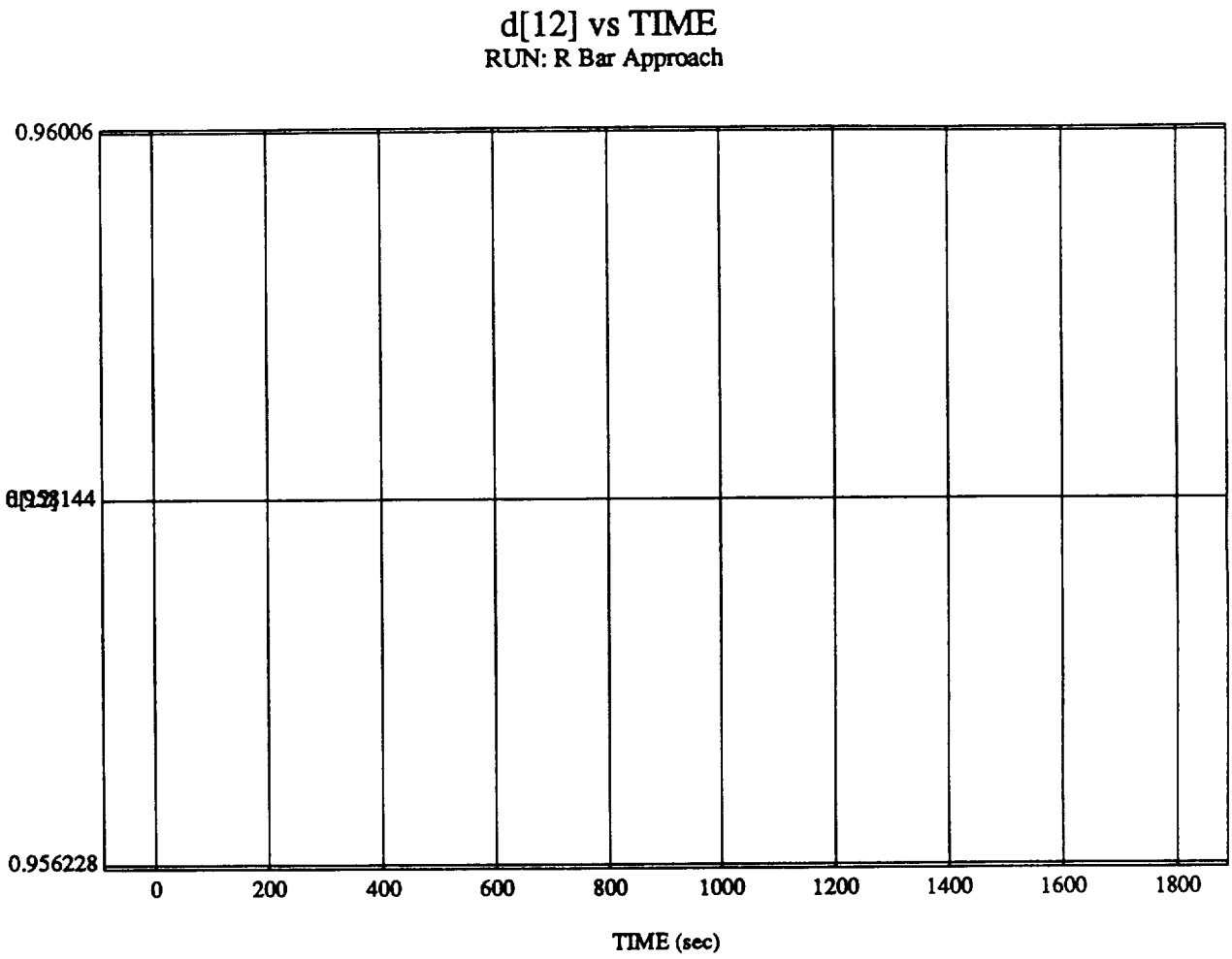
SIMULATION APPLICATION: ARIC Translational Controller Simulation



MODULE: ORBITER.lm_elev
DATA SAMPLING FREQUENCY: 0.200 Hz

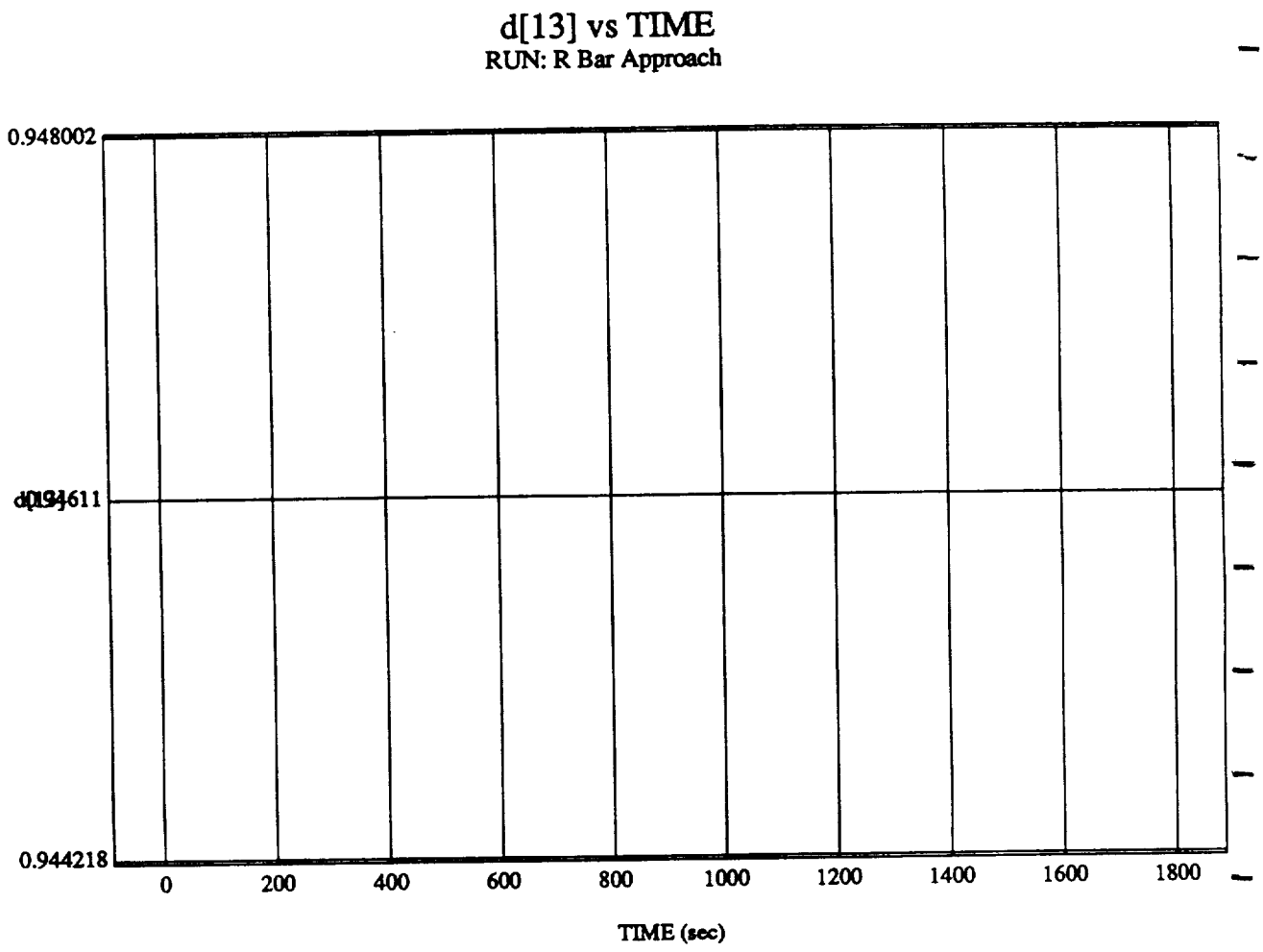


MODULE: ORBITER.lm_elev
DATA SAMPLING FREQUENCY: 0.200 Hz

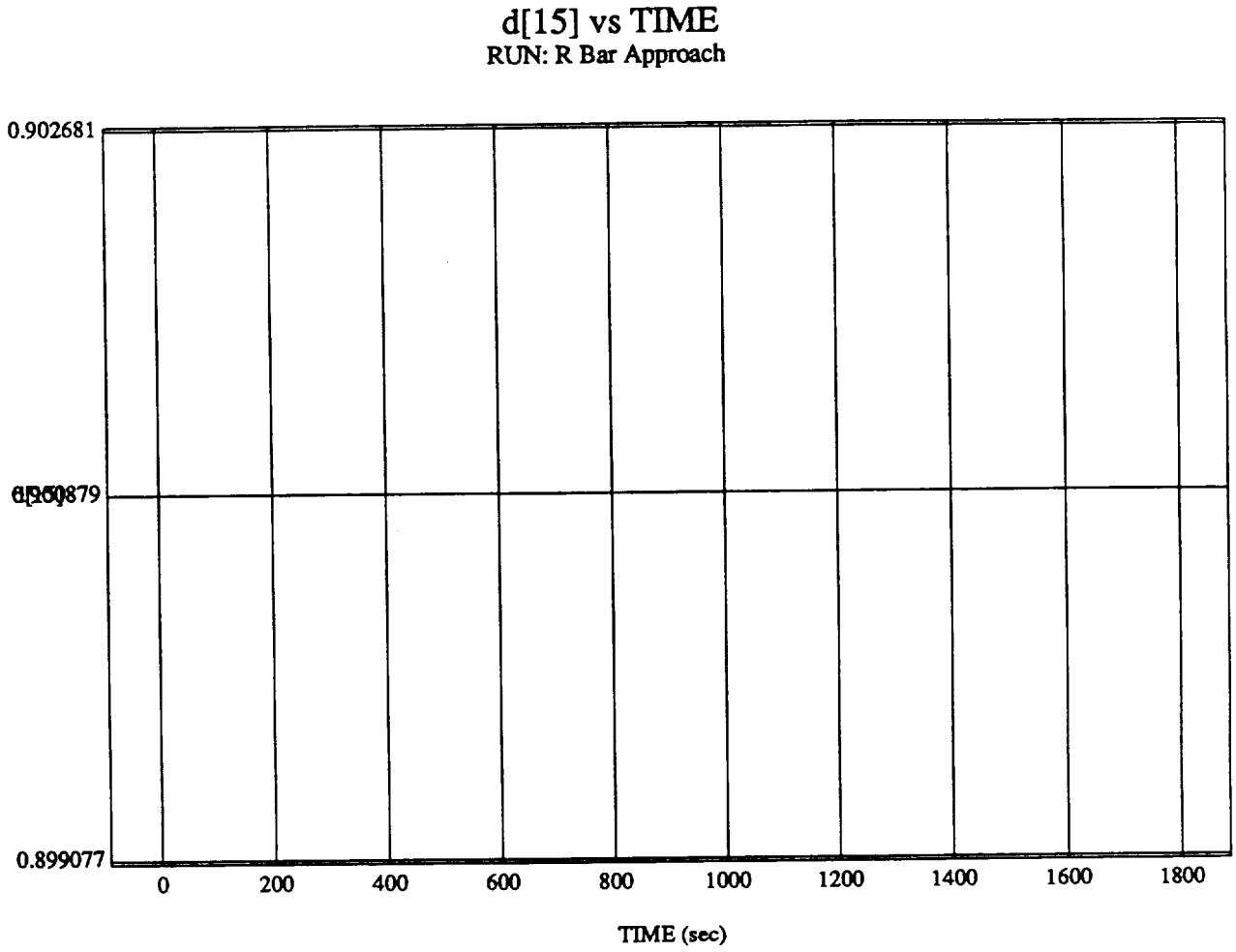


MODULE: ORBITER.lm_elev
DATA SAMPLING FREQUENCY: 0.200 Hz

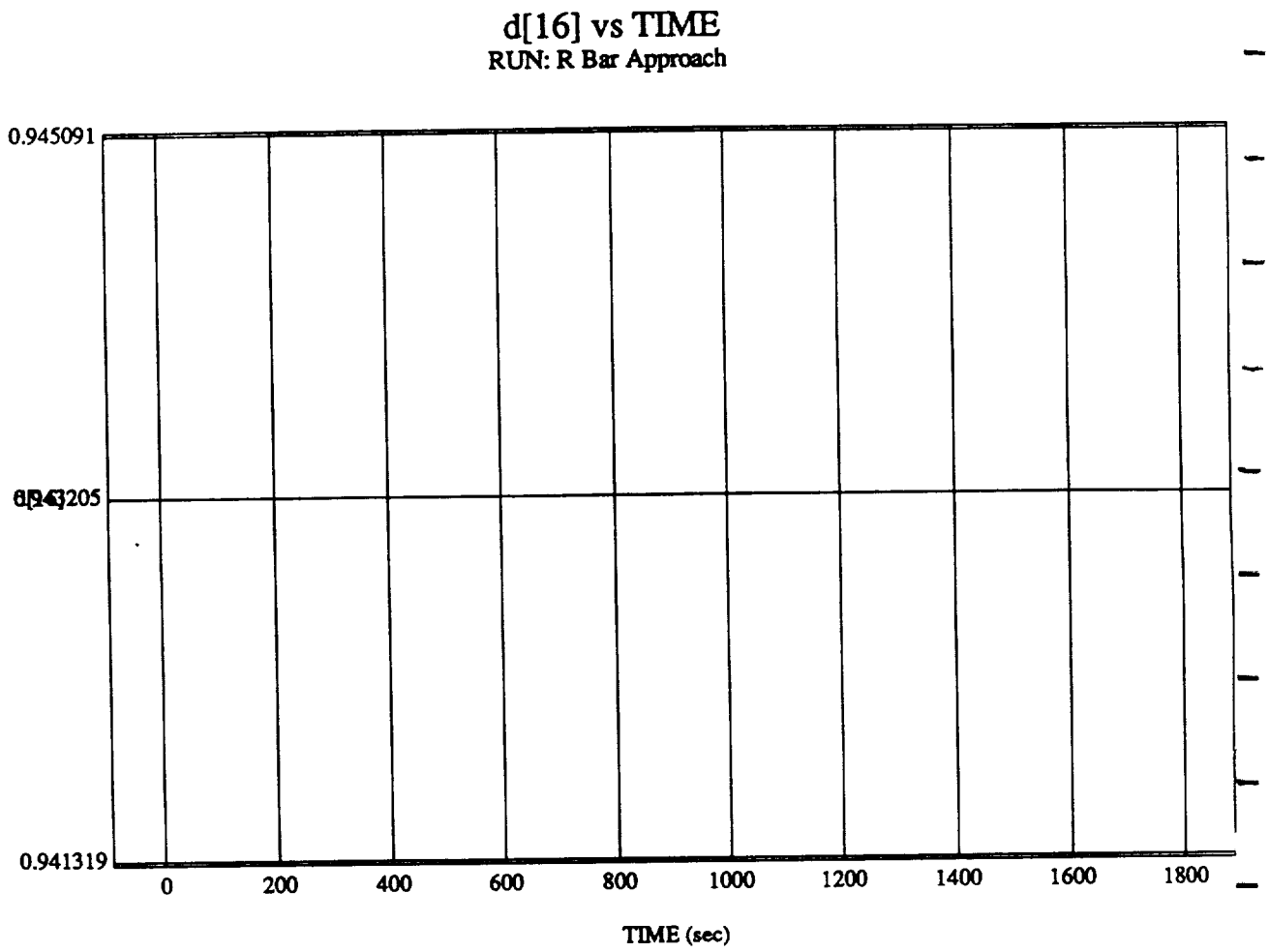
SIMULATION APPLICATION: ARIC Translational Controller Simulation



MODULE: ORBITER.lm_elev
DATA SAMPLING FREQUENCY: 0.200 Hz

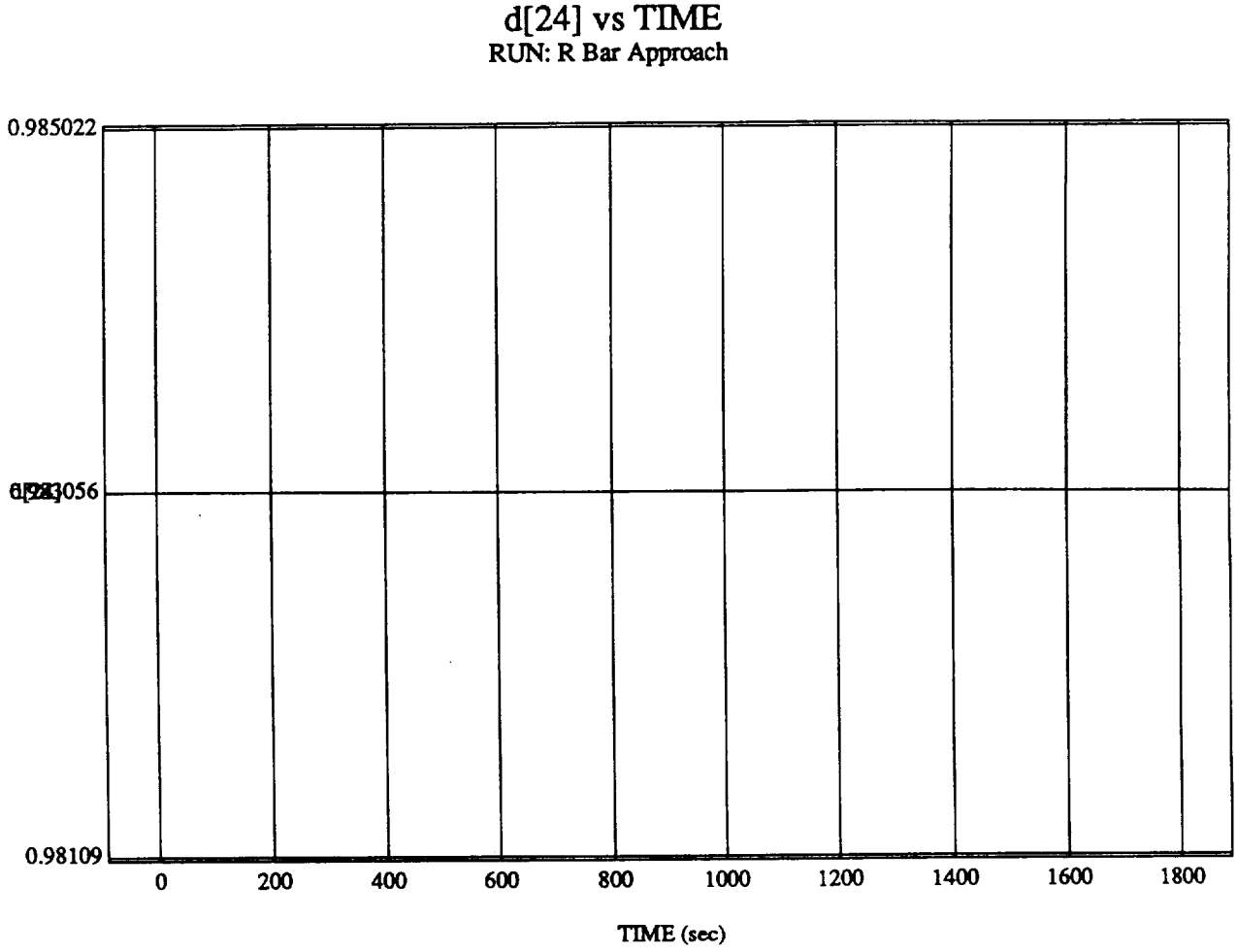


MODULE: ORBITER.lm_elev
DATA SAMPLING FREQUENCY: 0.200 Hz

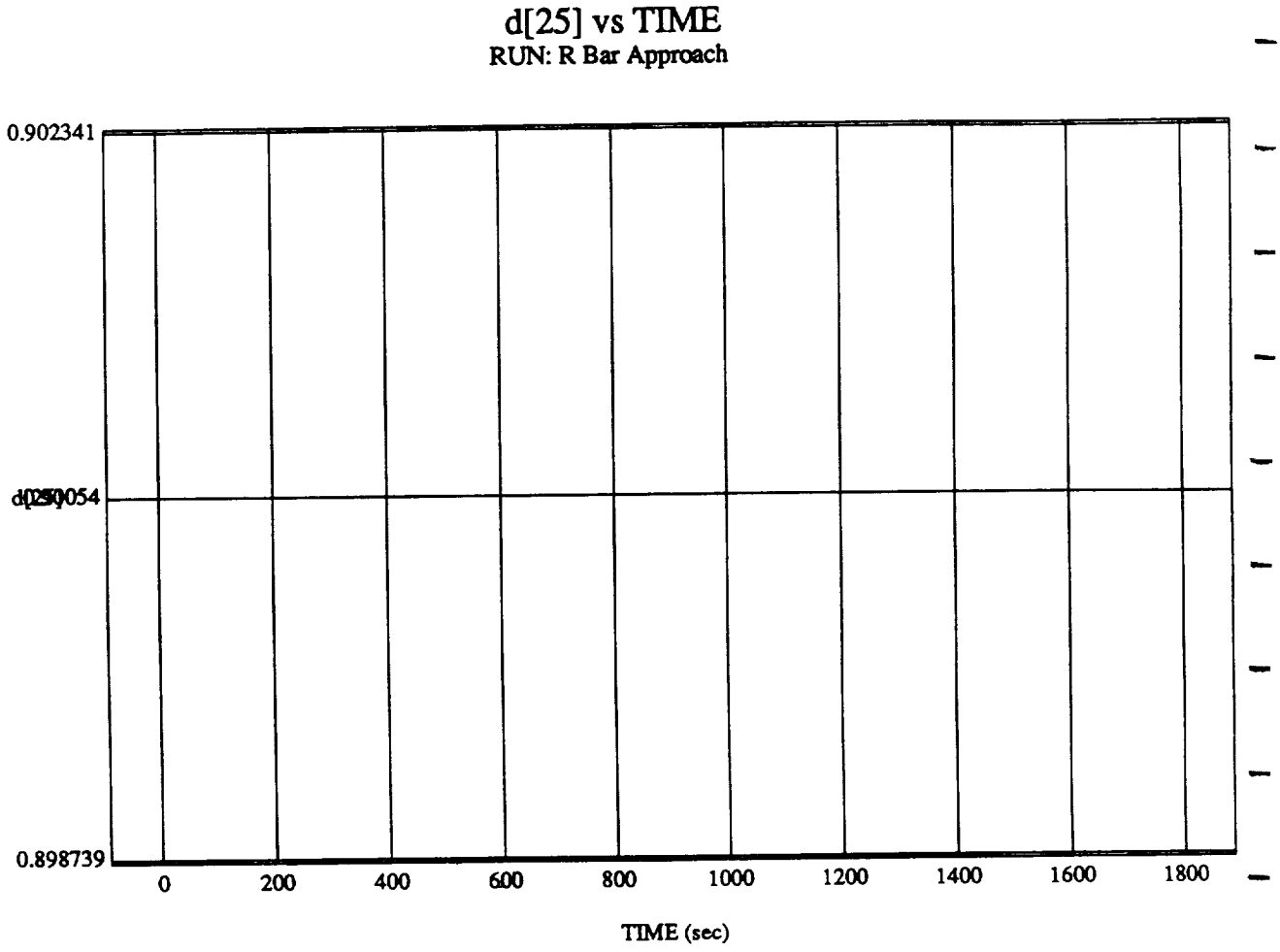


MODULE: ORBITER.lm_elev
DATA SAMPLING FREQUENCY: 0.200 Hz

SIMULATION APPLICATION: ARIC Translational Controller Simulation

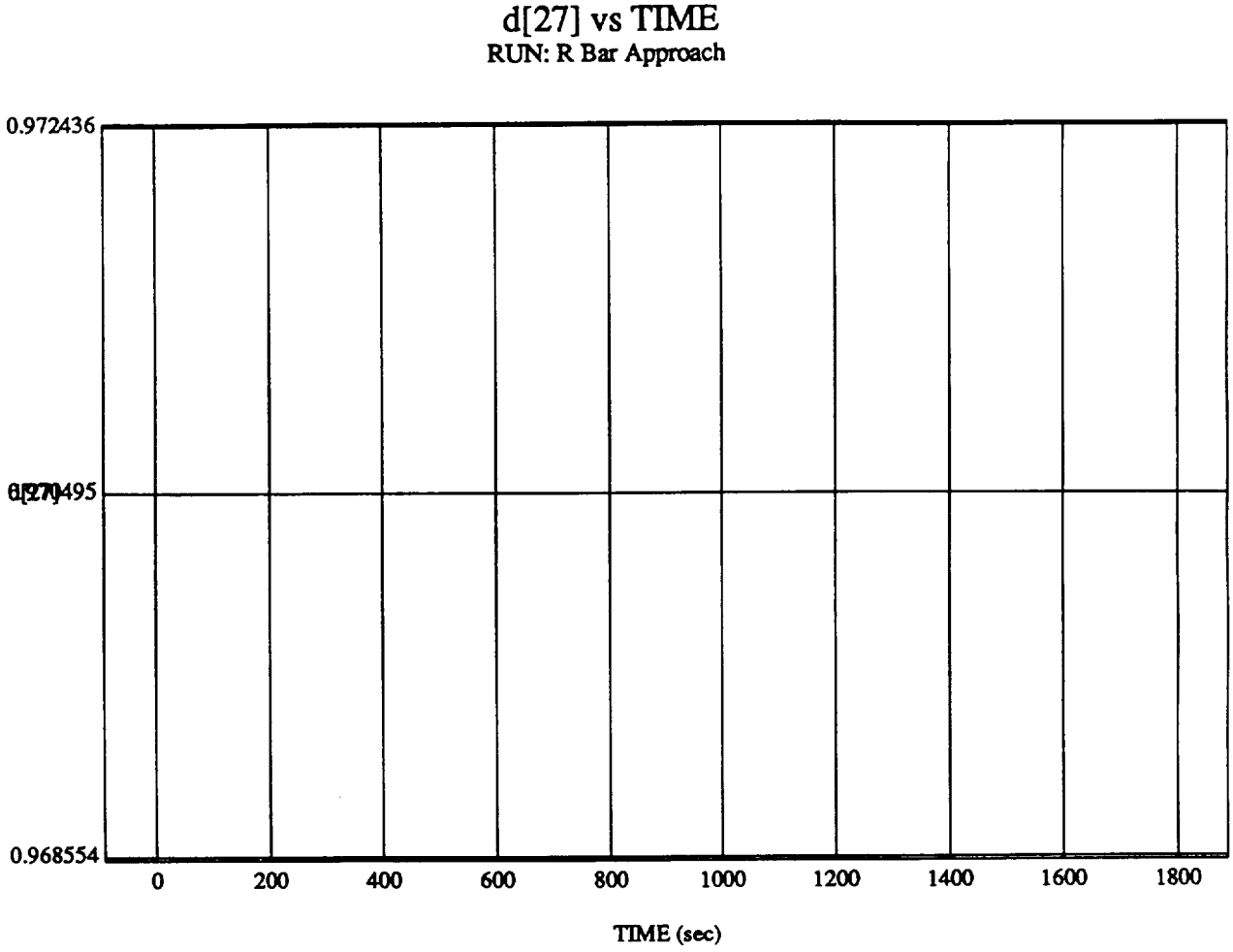


MODULE: ORBITER.lm_elev
DATA SAMPLING FREQUENCY: 0.200 Hz

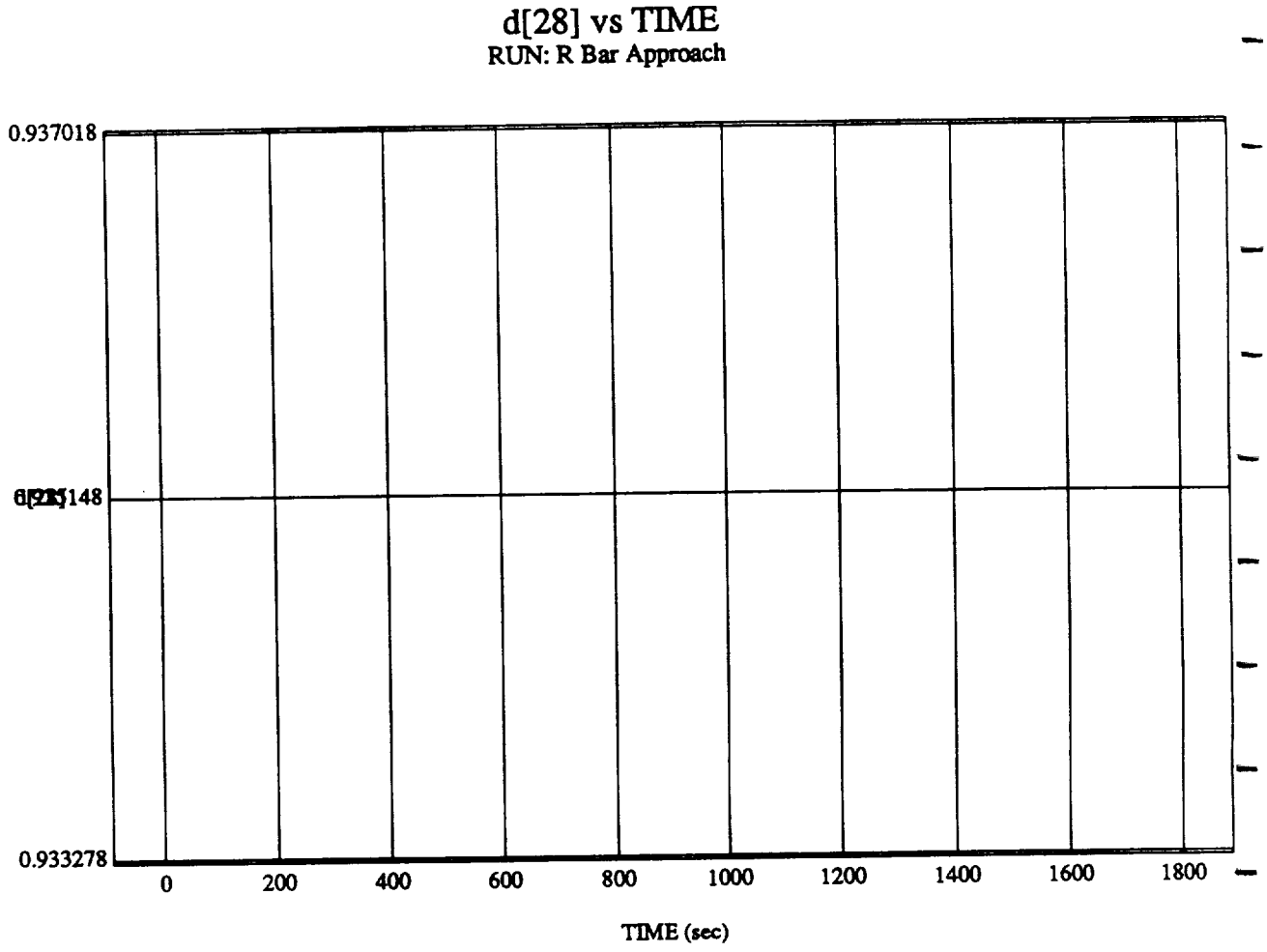


MODULE: ORBITER.lm_elev
DATA SAMPLING FREQUENCY: 0.200 Hz

SIMULATION APPLICATION: ARIC Translational Controller Simulation

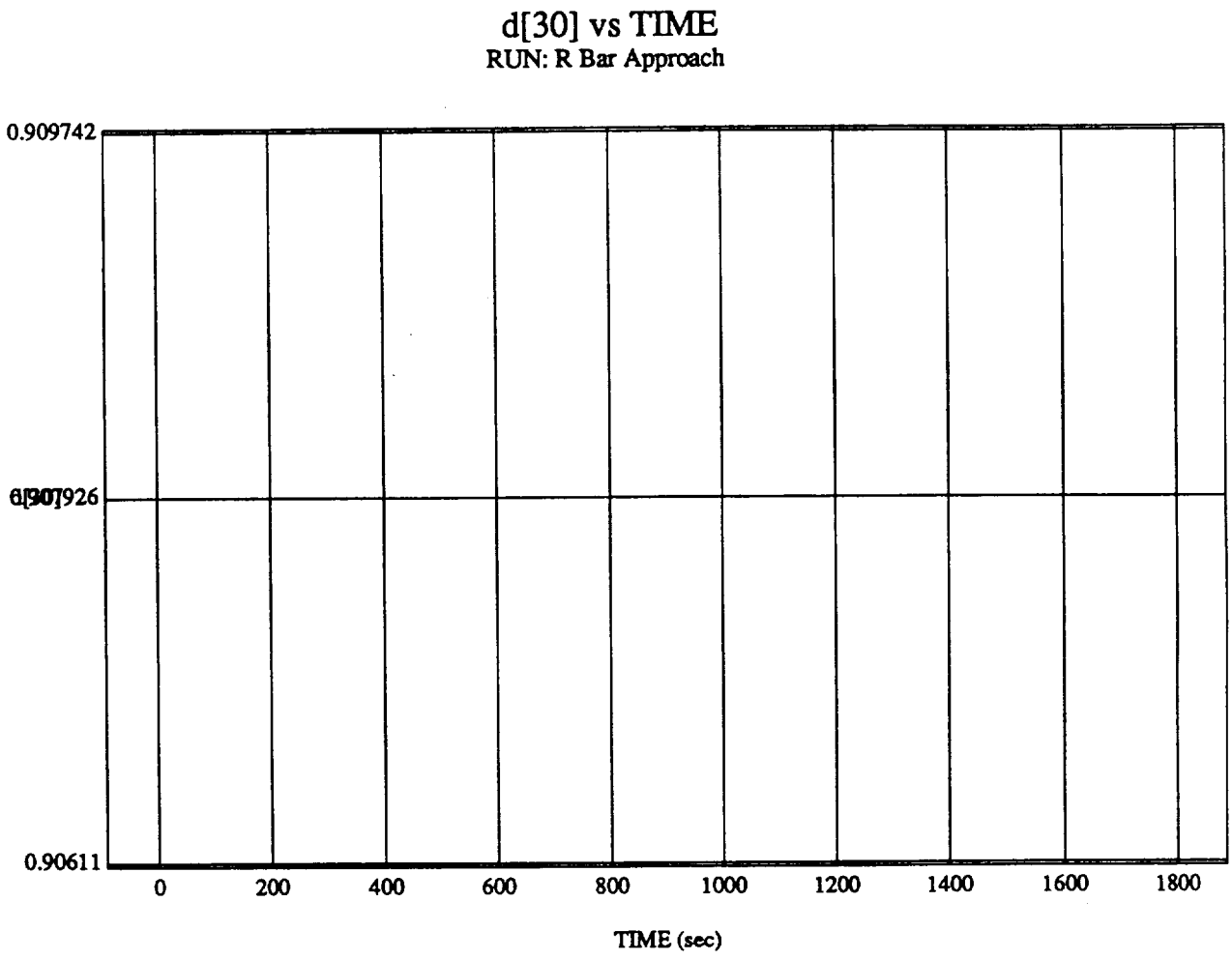


MODULE: ORBITER.lm_elev
DATA SAMPLING FREQUENCY: 0.200 Hz

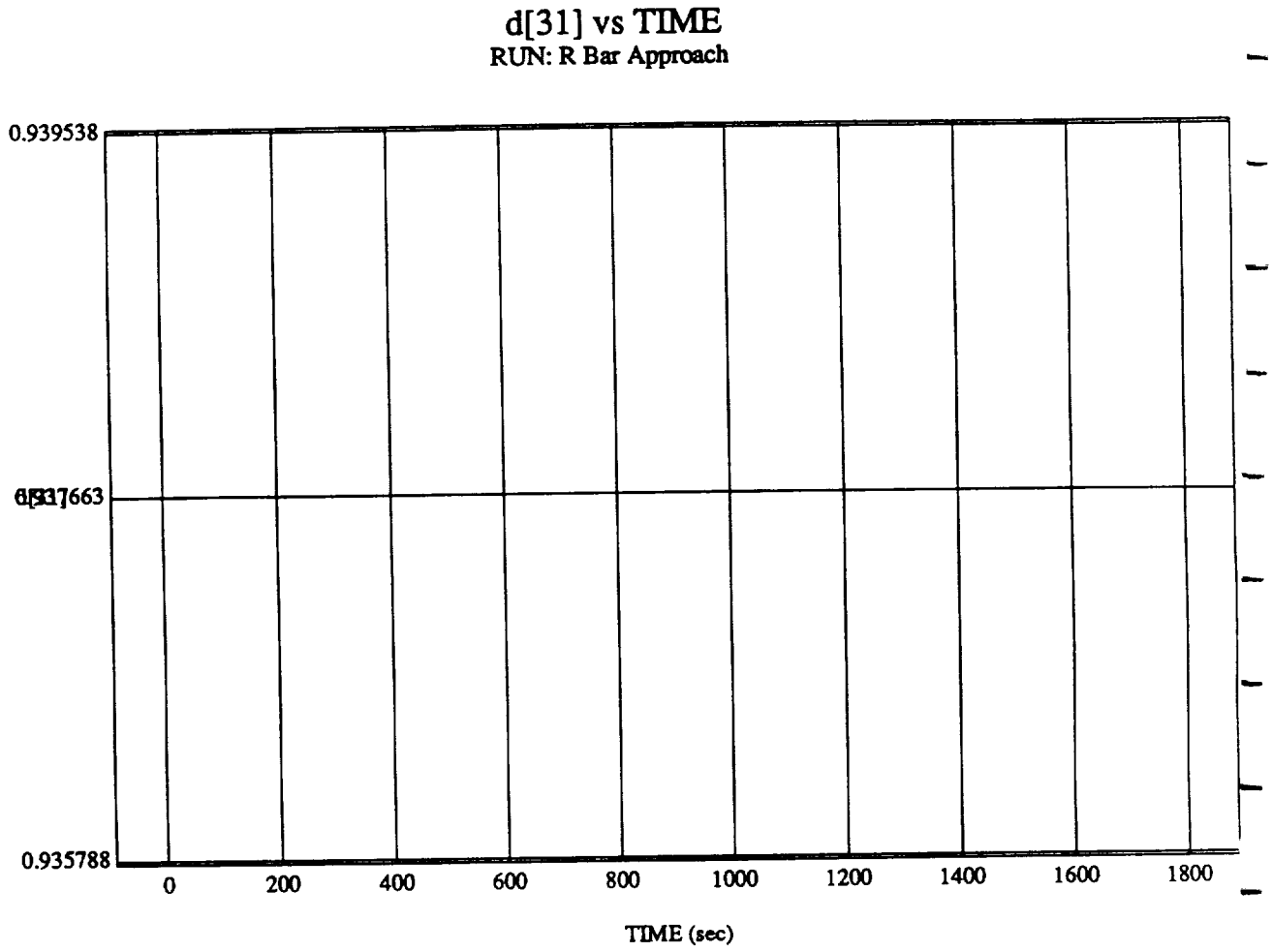


MODULE: ORBITER.lm_elev
DATA SAMPLING FREQUENCY: 0.200 Hz

SIMULATION APPLICATION: ARIC Translational Controller Simulation

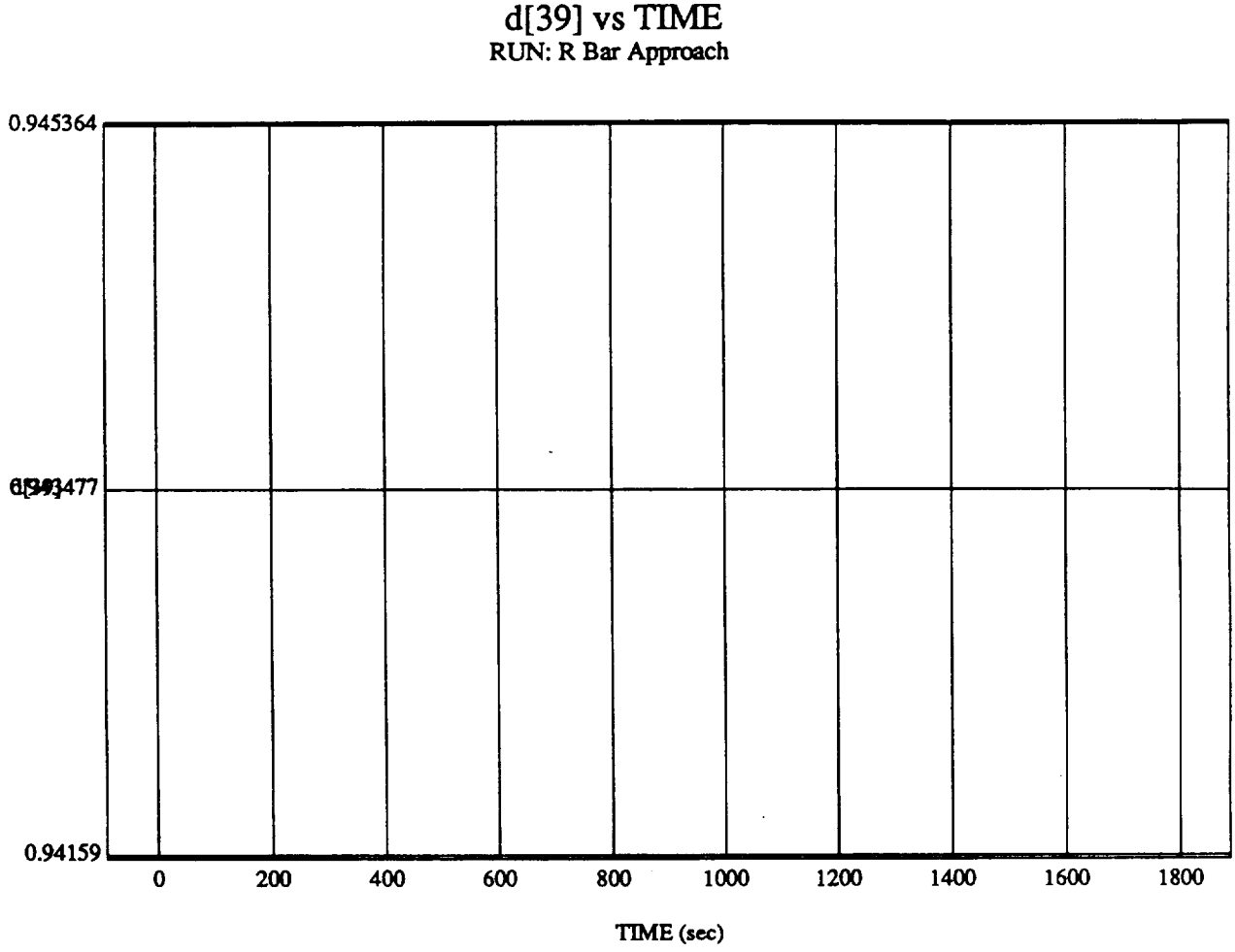


MODULE: ORBITER.lm_elev
DATA SAMPLING FREQUENCY: 0.200 Hz

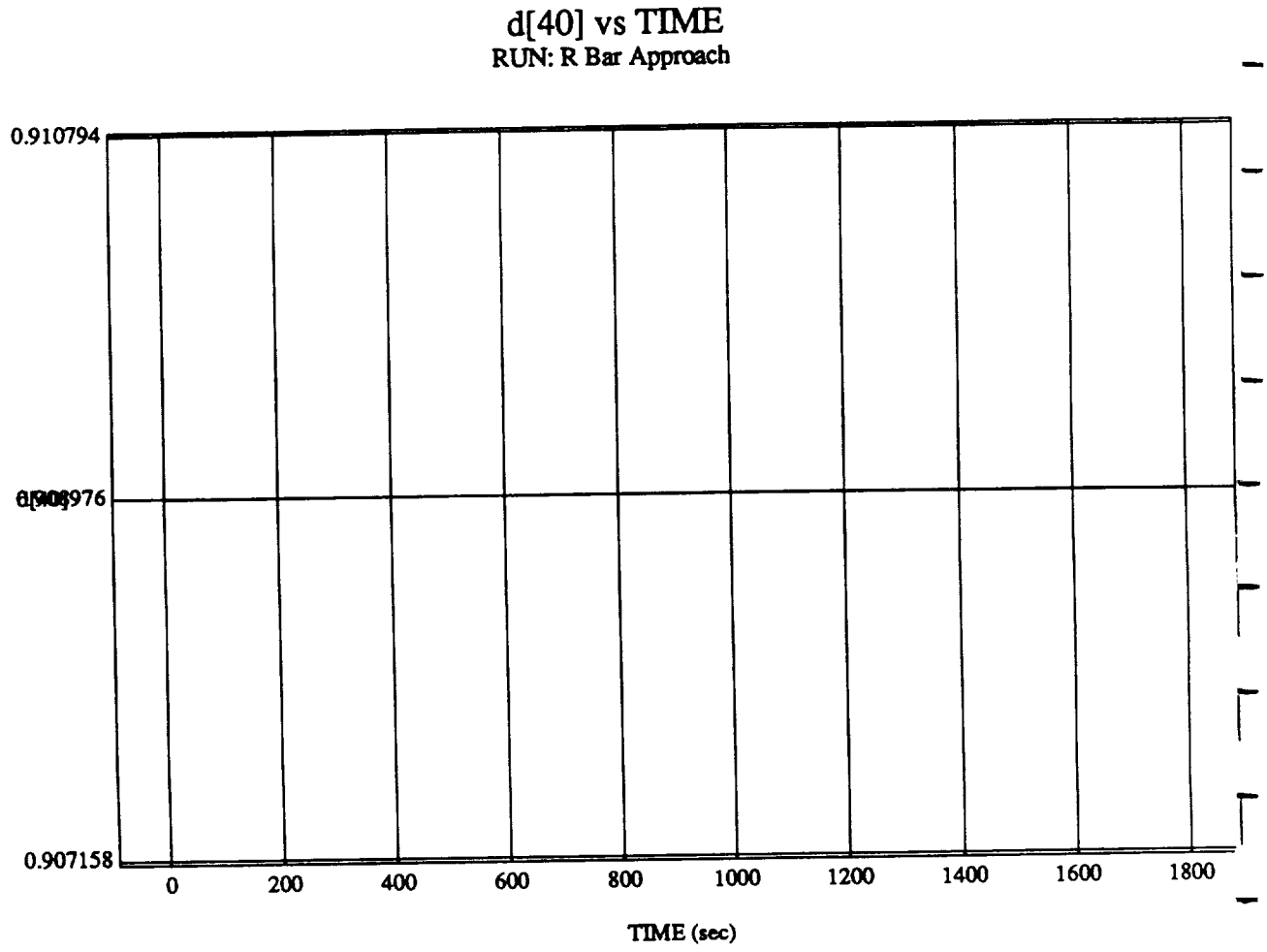


MODULE: ORBITER.lm_elev
DATA SAMPLING FREQUENCY: 0.200 Hz

SIMULATION APPLICATION: ARIC Translational Controller Simulation

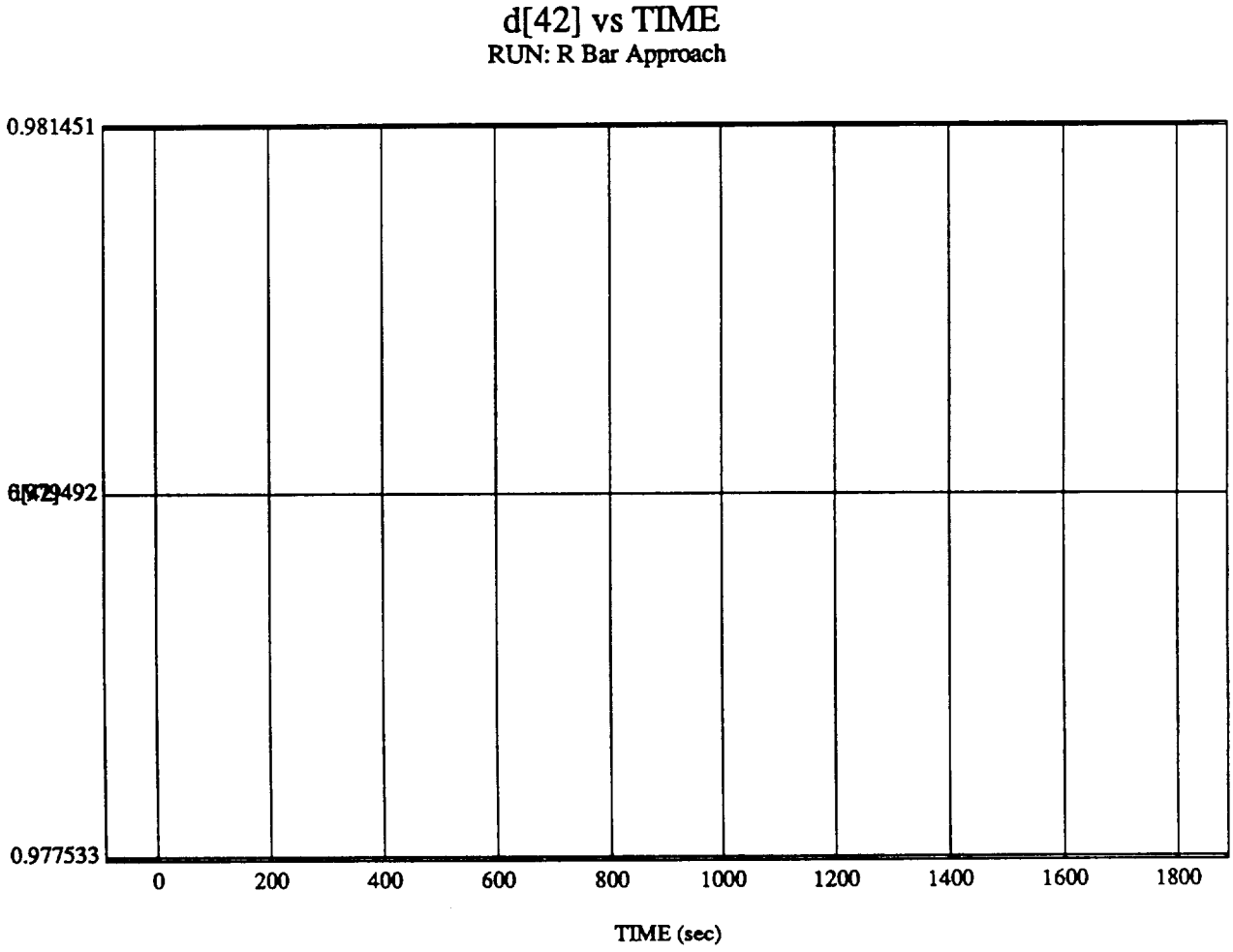


MODULE: ORBITER.lm_elev
DATA SAMPLING FREQUENCY: 0.200 Hz

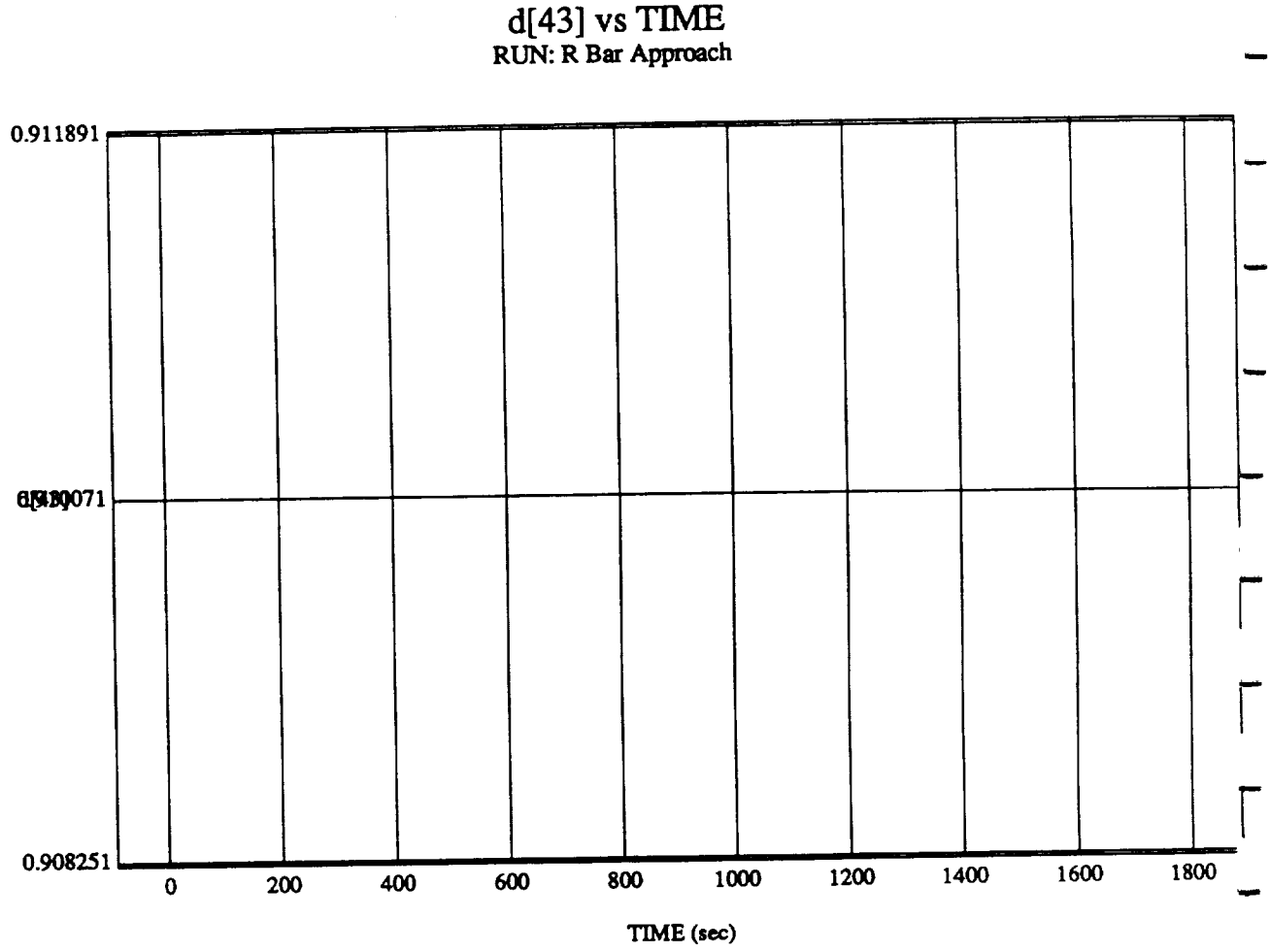


MODULE: ORBITER_lm_elev
DATA SAMPLING FREQUENCY: 0.200 Hz

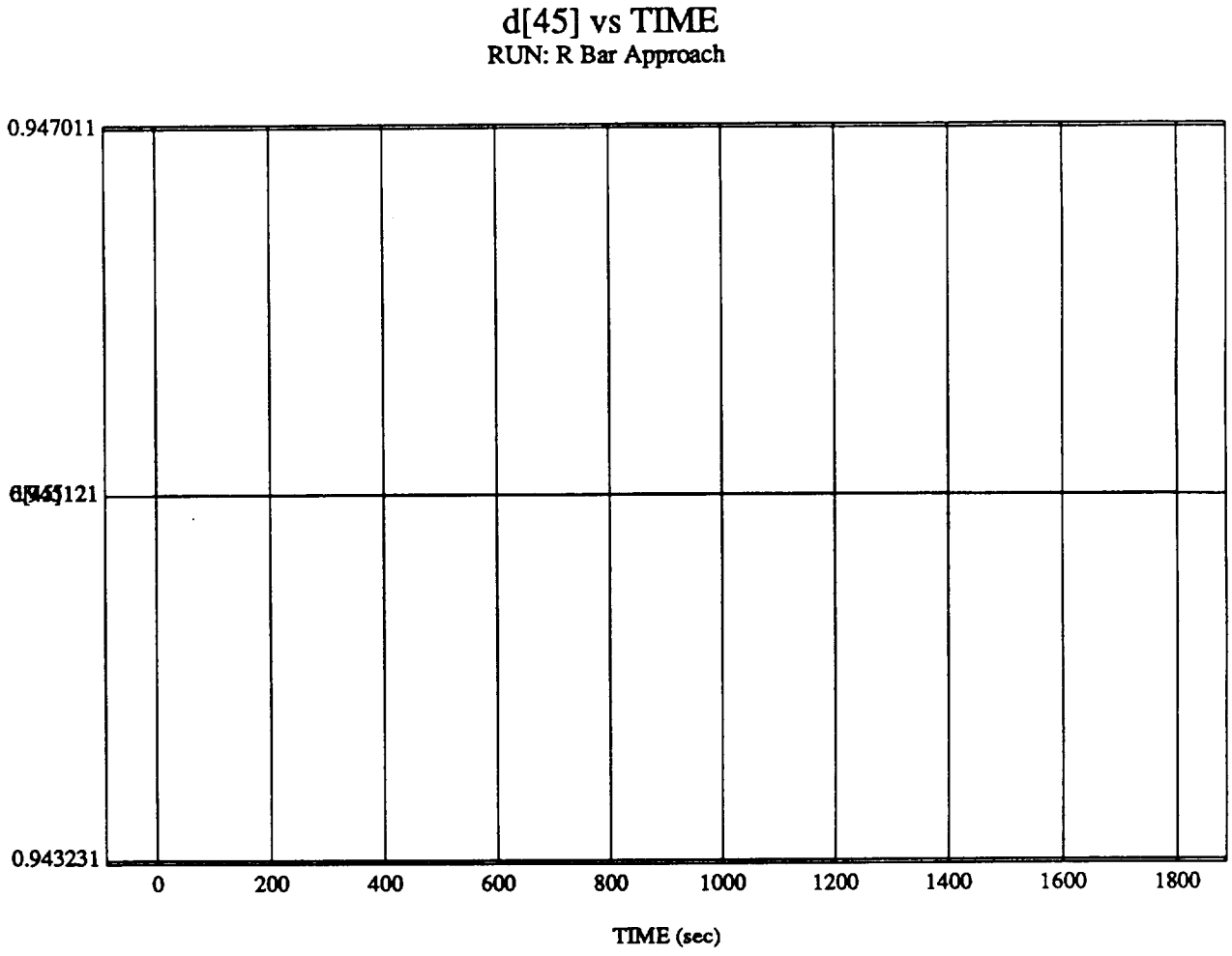
SIMULATION APPLICATION: ARIC Translational Controller Simulation



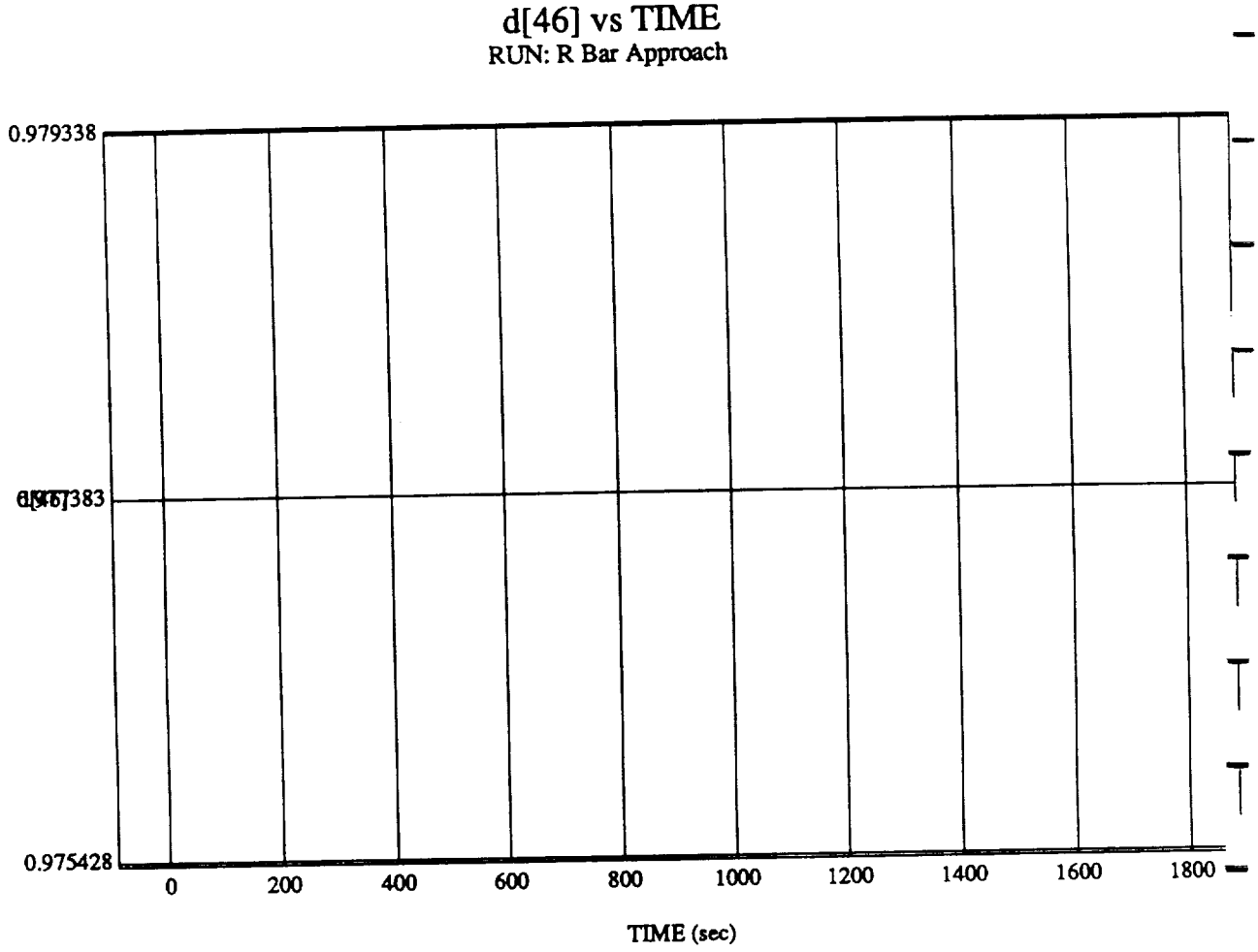
MODULE: ORBITER.lm_elev
DATA SAMPLING FREQUENCY: 0.200 Hz



MODULE: ORBITER.lm_elev
DATA SAMPLING FREQUENCY: 0.200 Hz



MODULE: ORBITER.lm_elev
DATA SAMPLING FREQUENCY: 0.200 Hz

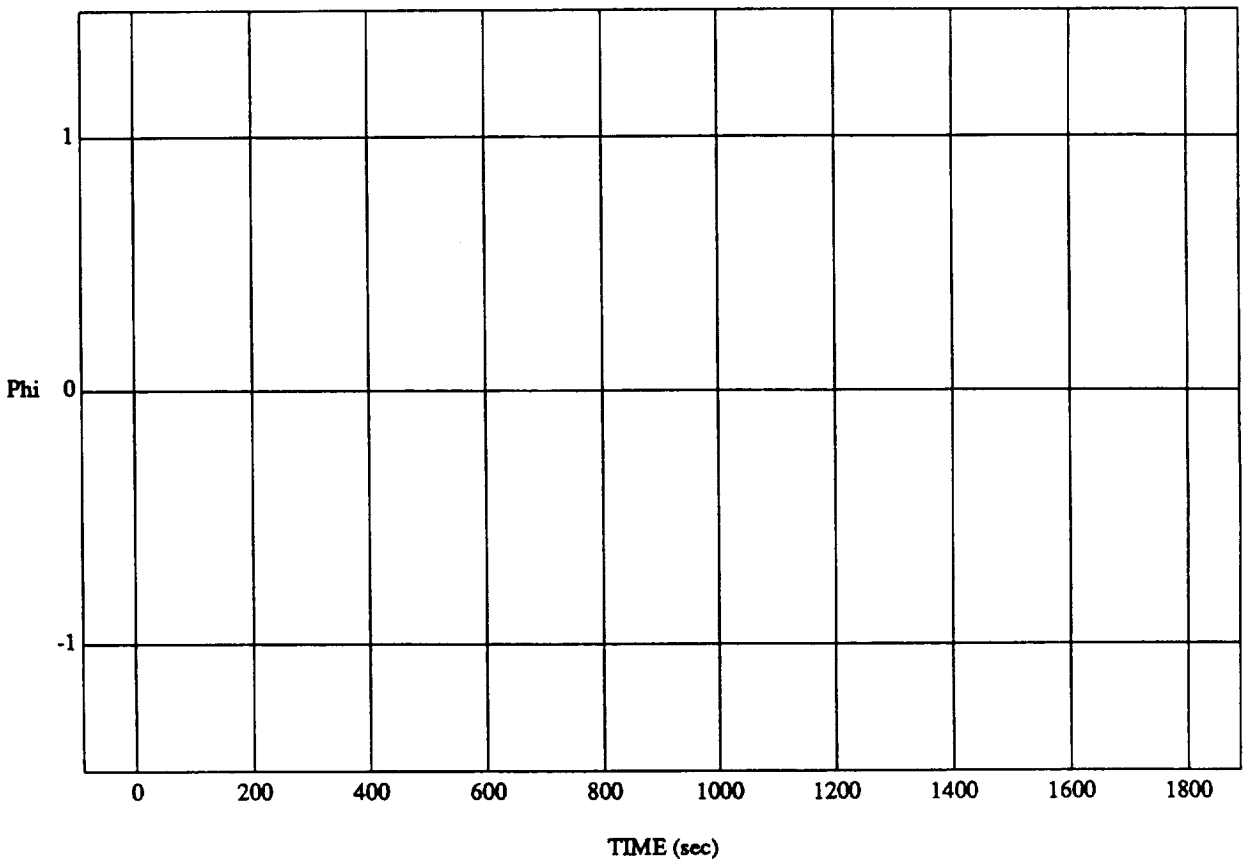


MODULE: ORBITER.lm_elev
DATA SAMPLING FREQUENCY: 0.200 Hz

SIMULATION APPLICATION: ARIC Translational Controller Simulation

Phi vs TIME

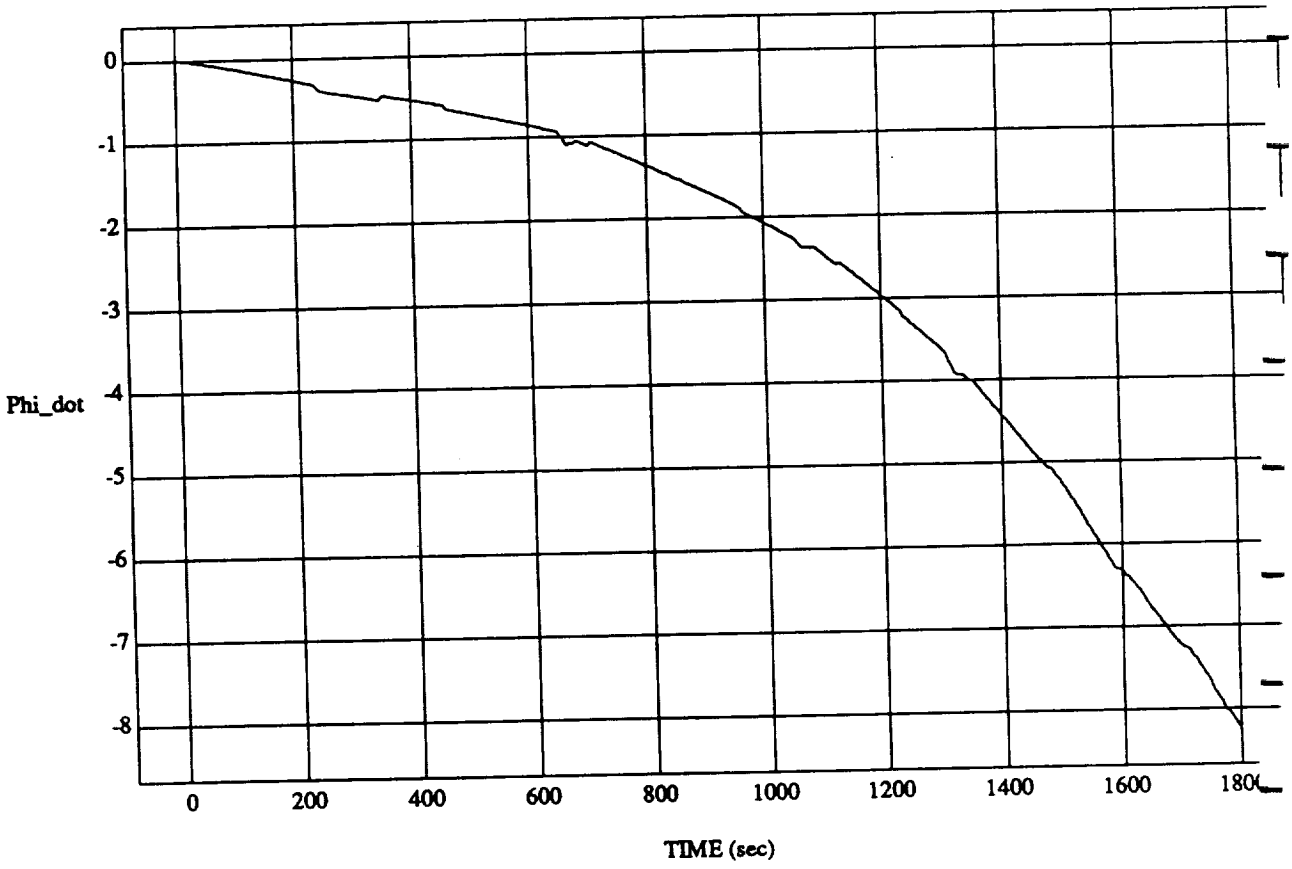
RUN: R Bar Approach



MODULE: ORBITER.lm_range
DATA SAMPLING FREQUENCY: 0.200 Hz

Phi_dot vs TIME

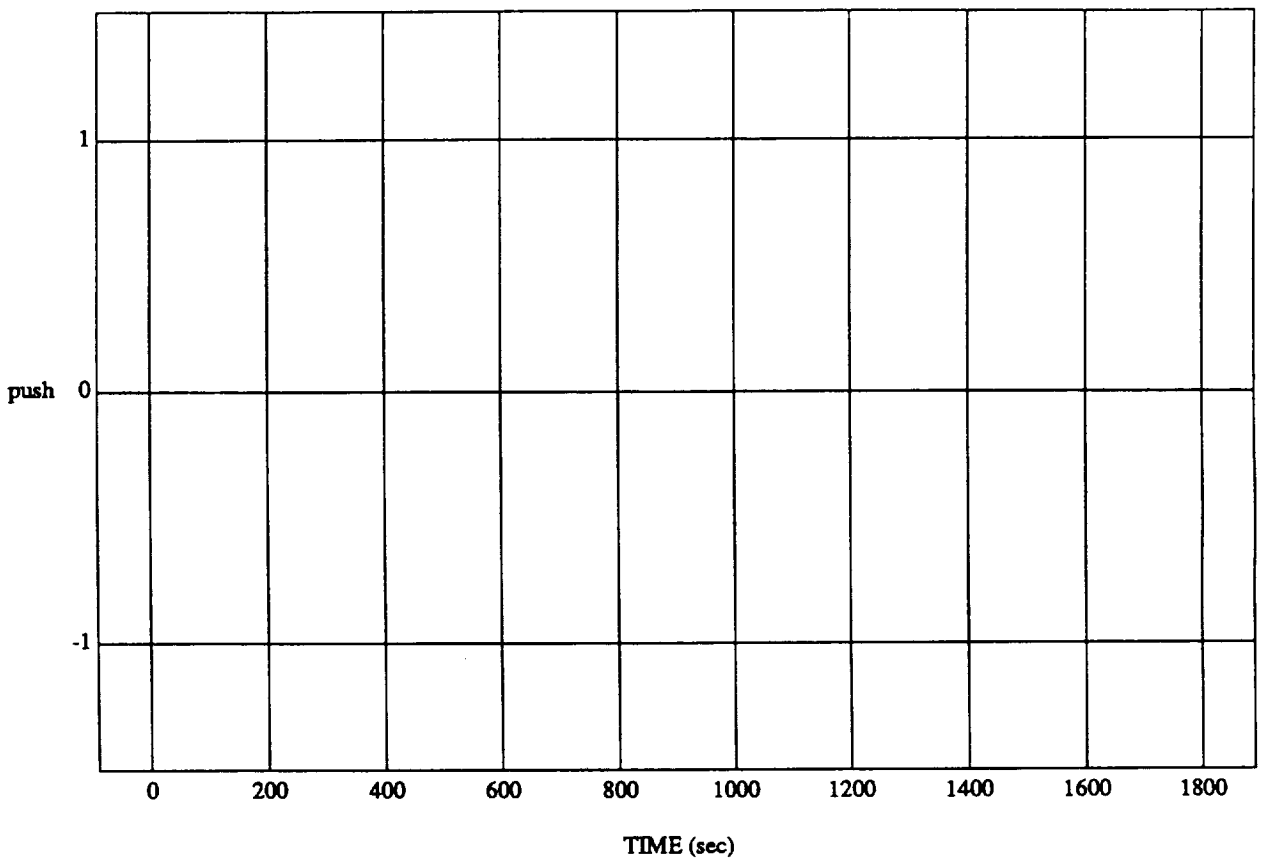
RUN: R Bar Approach



MODULE: ORBITER.lm_range
DATA SAMPLING FREQUENCY: 0.200 Hz

push vs TIME

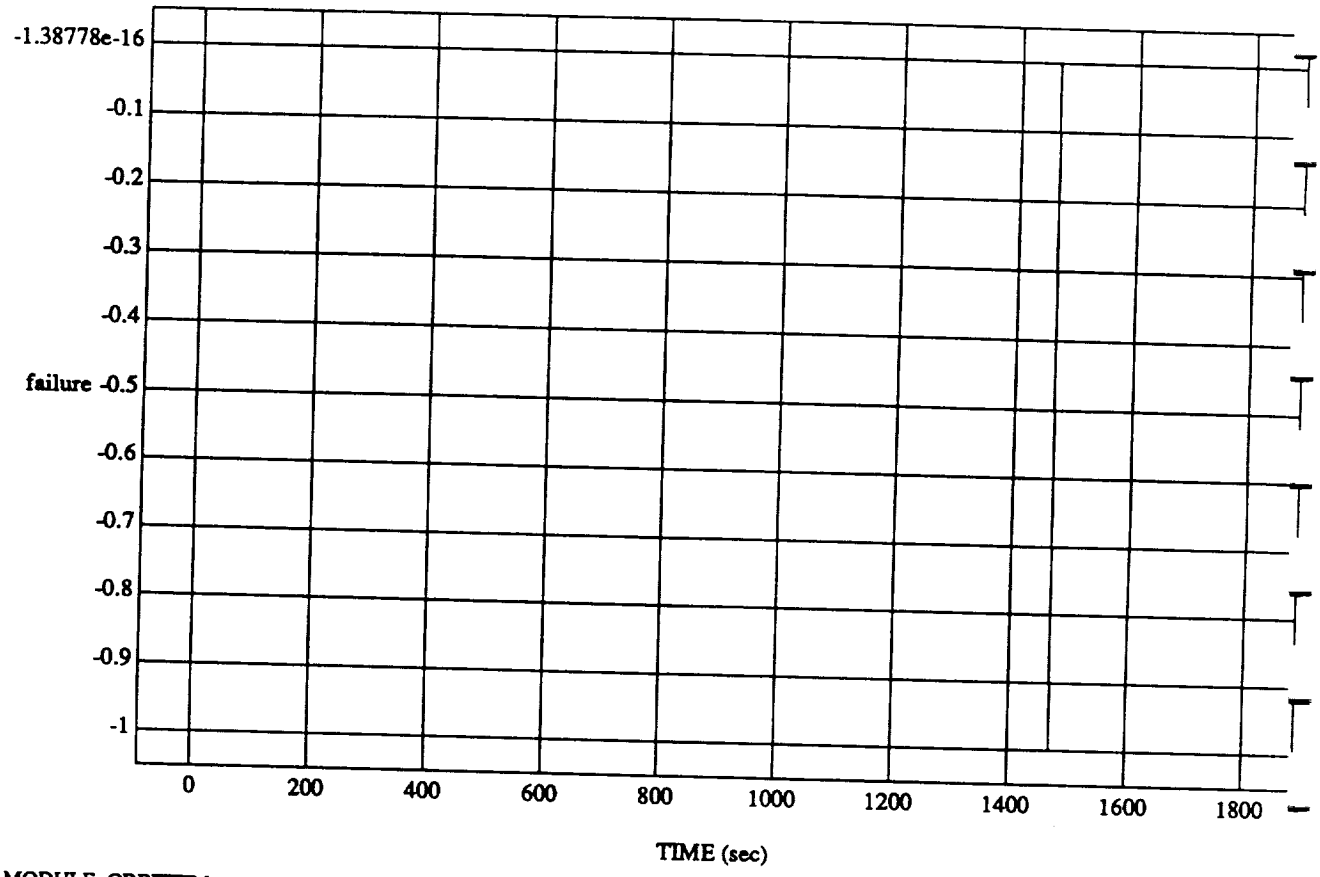
RUN: R Bar Approach



MODULE: ORBITER.lm_range
DATA SAMPLING FREQUENCY: 0.200 Hz

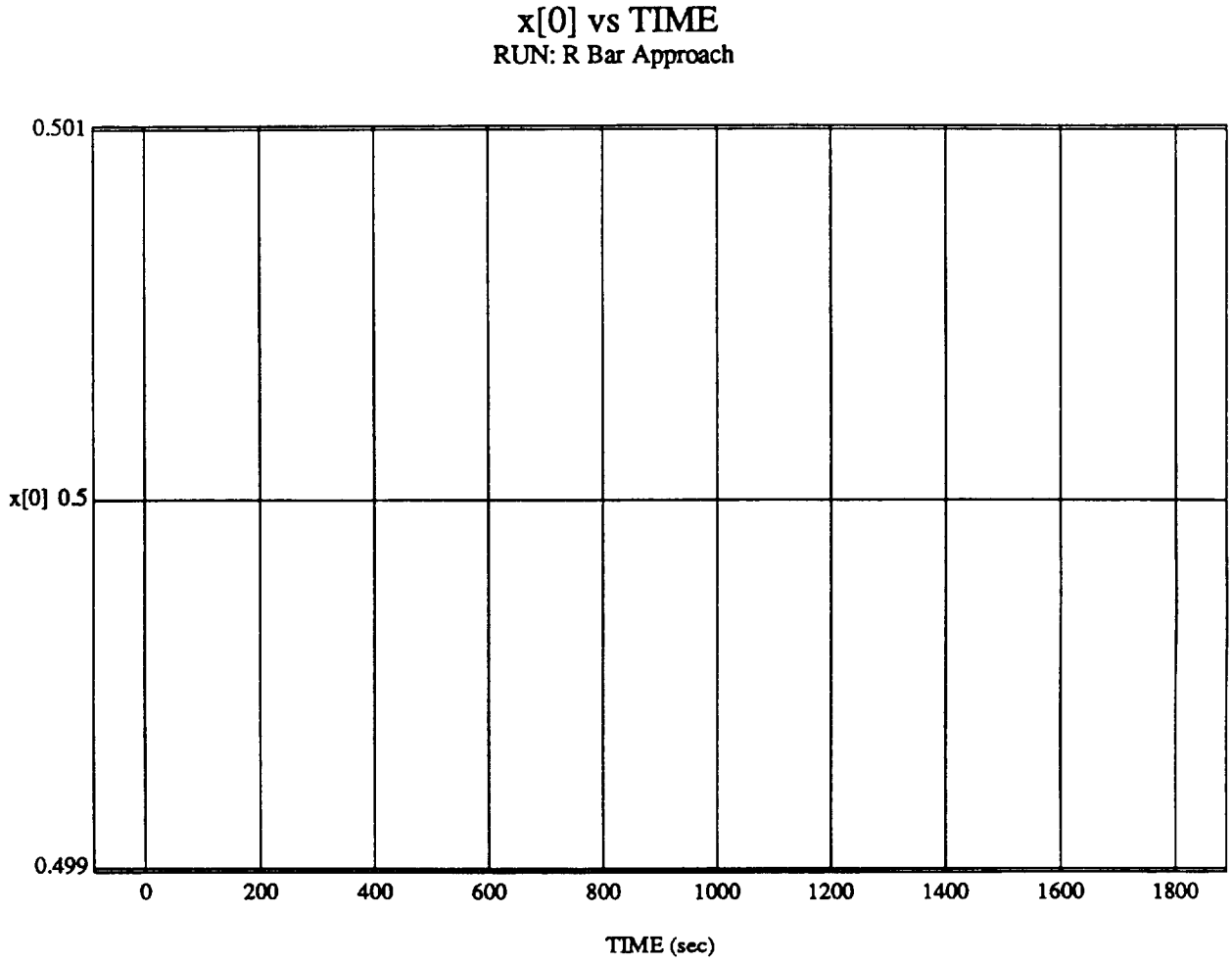
failure vs TIME

RUN: R Bar Approach



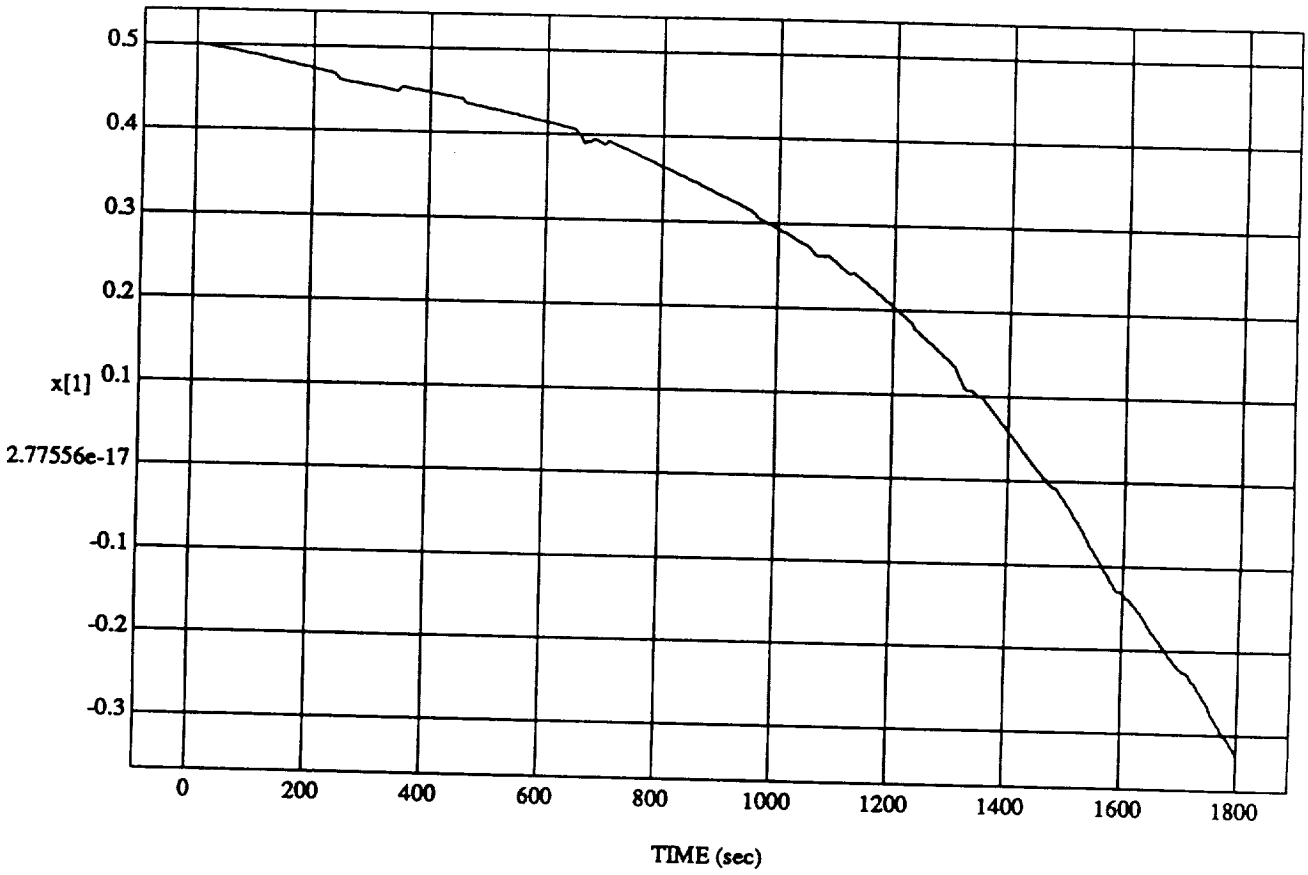
MODULE: ORBITER.lm_range
DATA SAMPLING FREQUENCY: 0.200 Hz

SIMULATION APPLICATION: ARIC Translational Controller Simulation

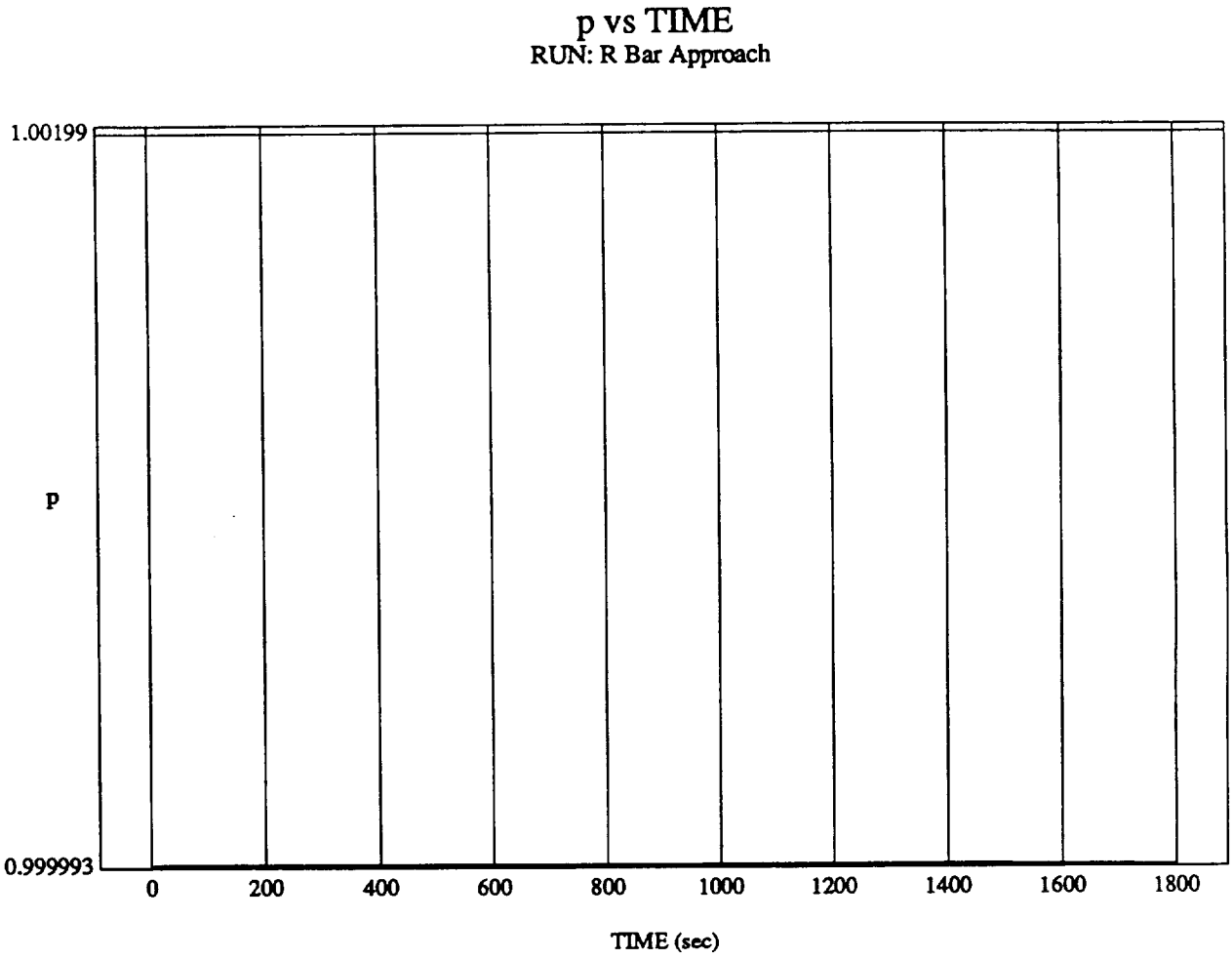


MODULE: ORBITER.lm_range
DATA SAMPLING FREQUENCY: 0.200 Hz

x[1] vs TIME
RUN: R Bar Approach



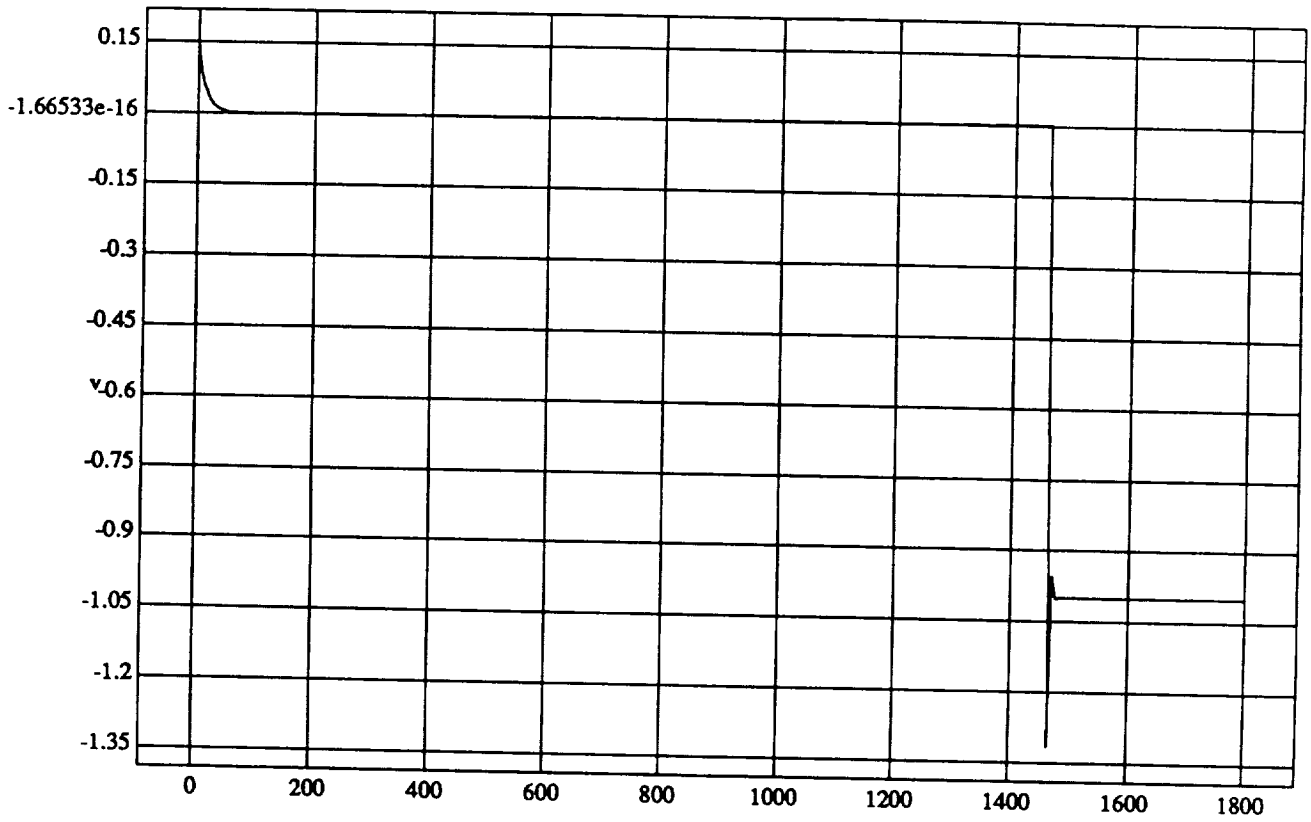
MODULE: ORBITER.lm_range
DATA SAMPLING FREQUENCY: 0.200 Hz



MODULE: ORBITER.lm_range
DATA SAMPLING FREQUENCY: 0.200 Hz

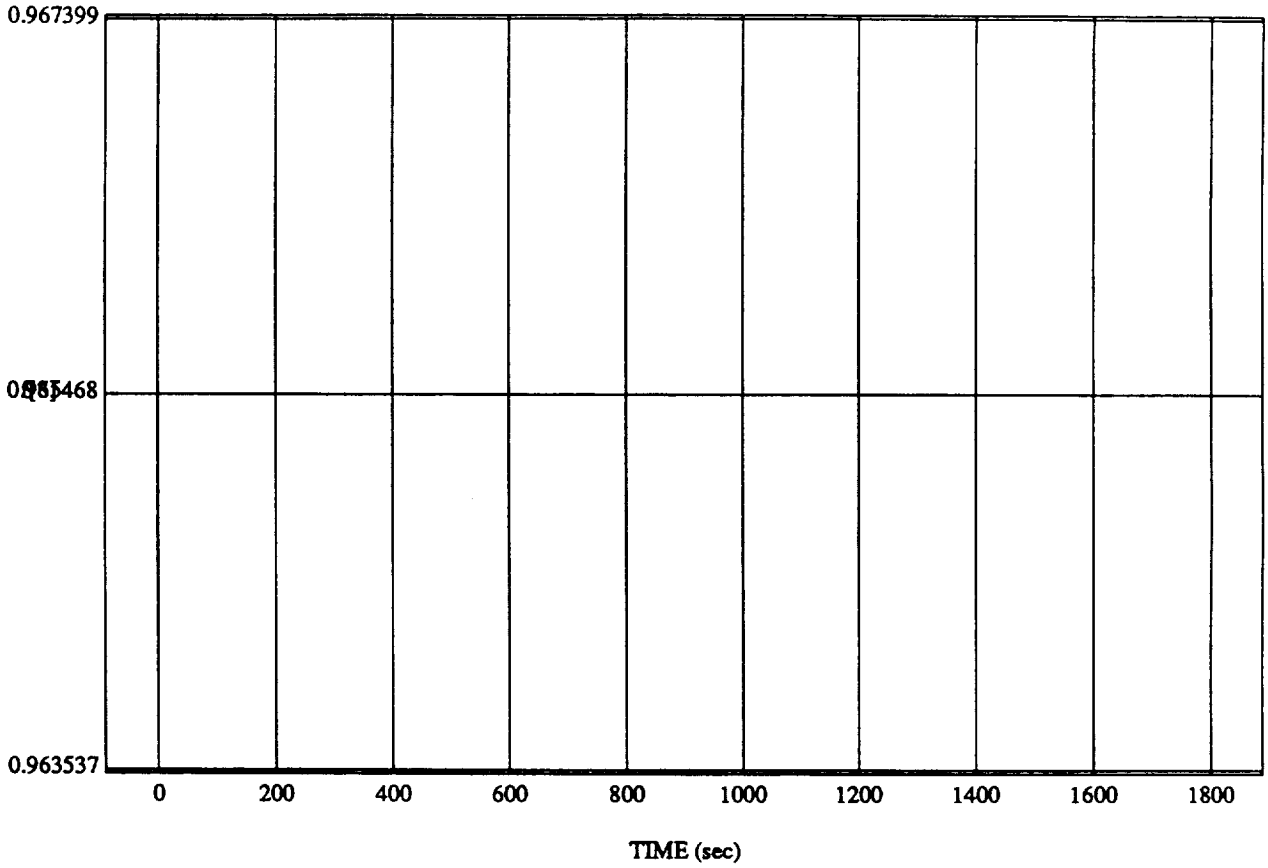
SIMULATION APPLICATION: ARIC Translational Controller Simulation

v vs TIME
RUN: R Bar Approach

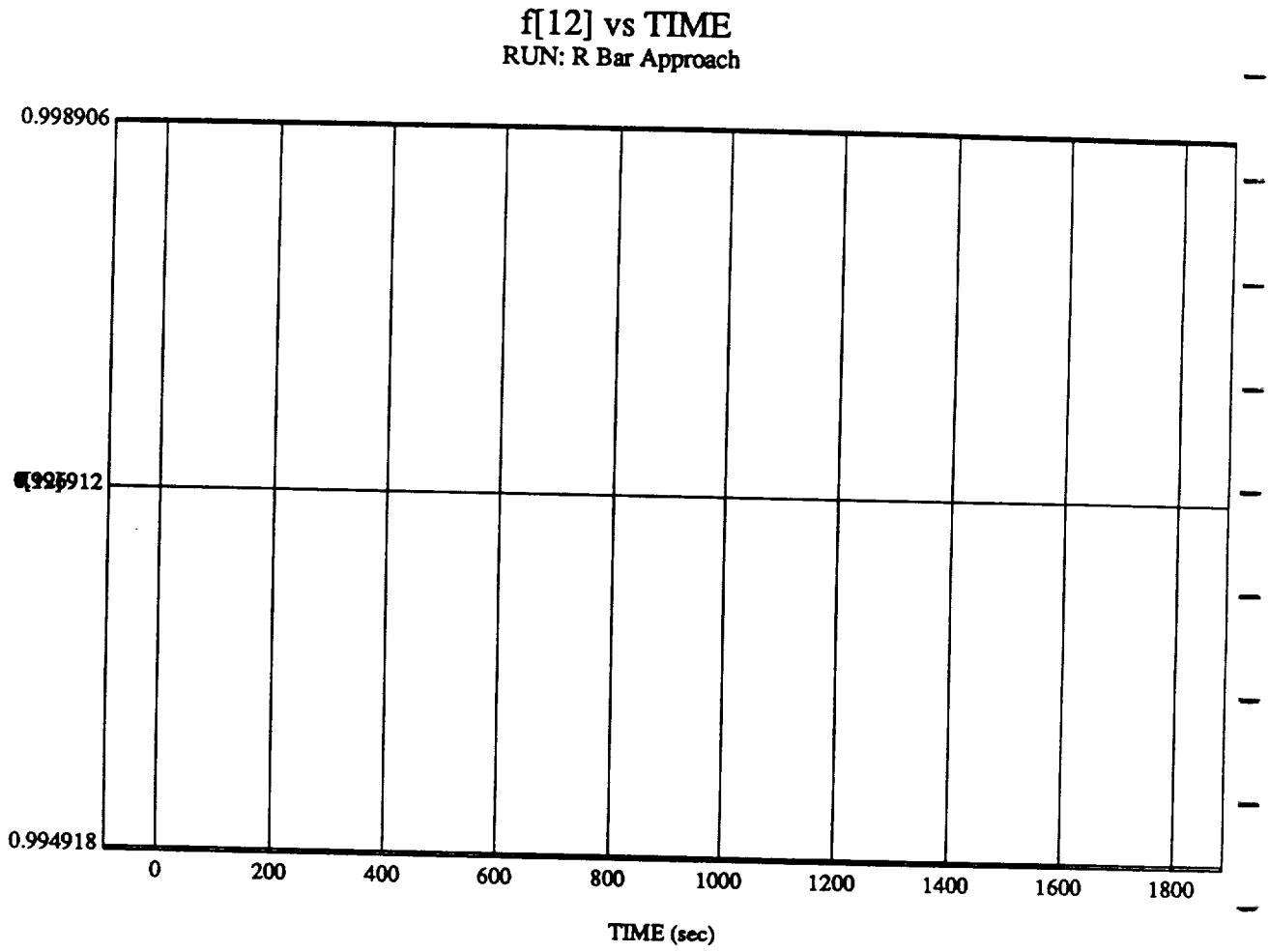


MODULE: ORBITER.lm_range
DATA SAMPLING FREQUENCY: 0.200 Hz

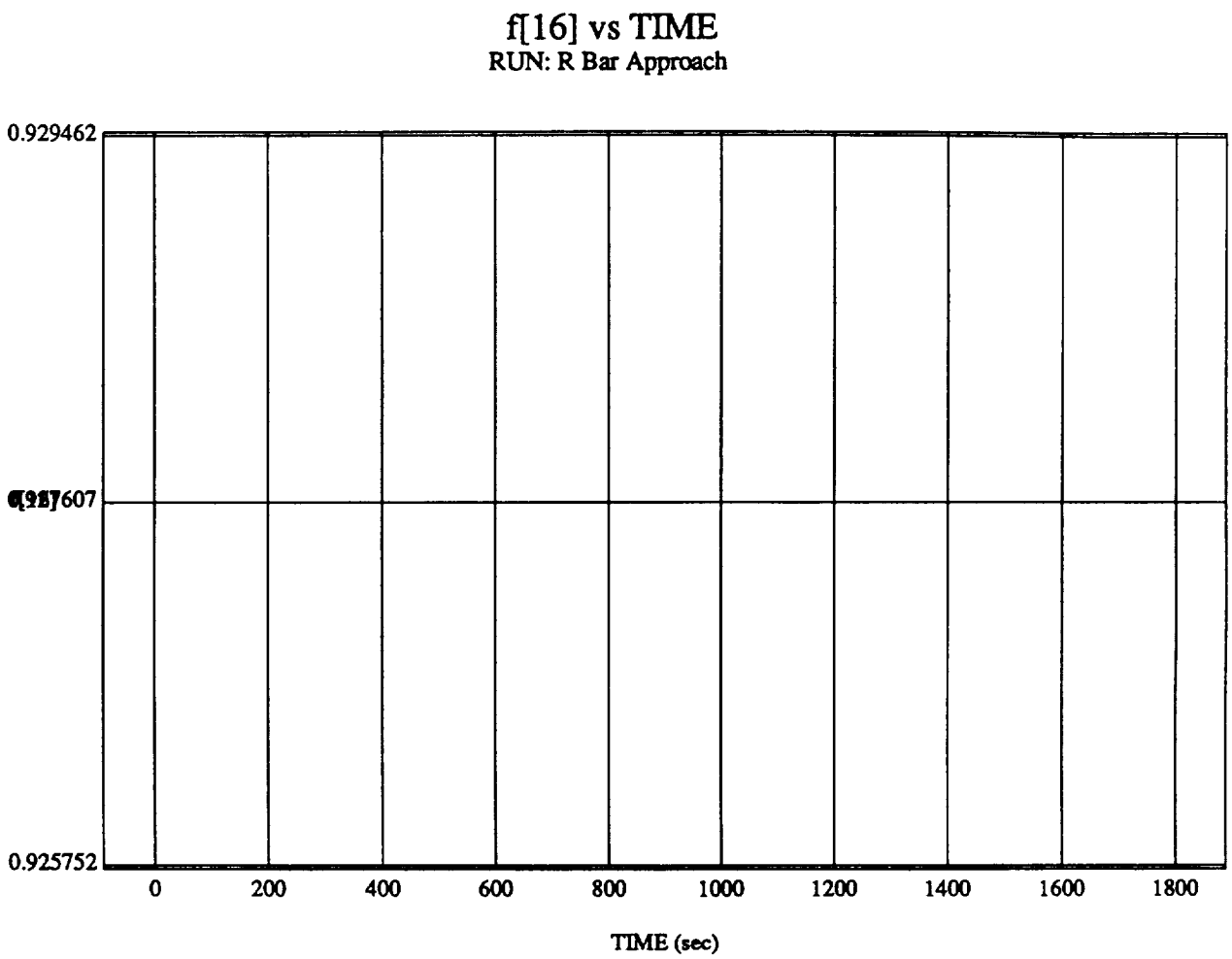
f[8] vs TIME
RUN: R Bar Approach



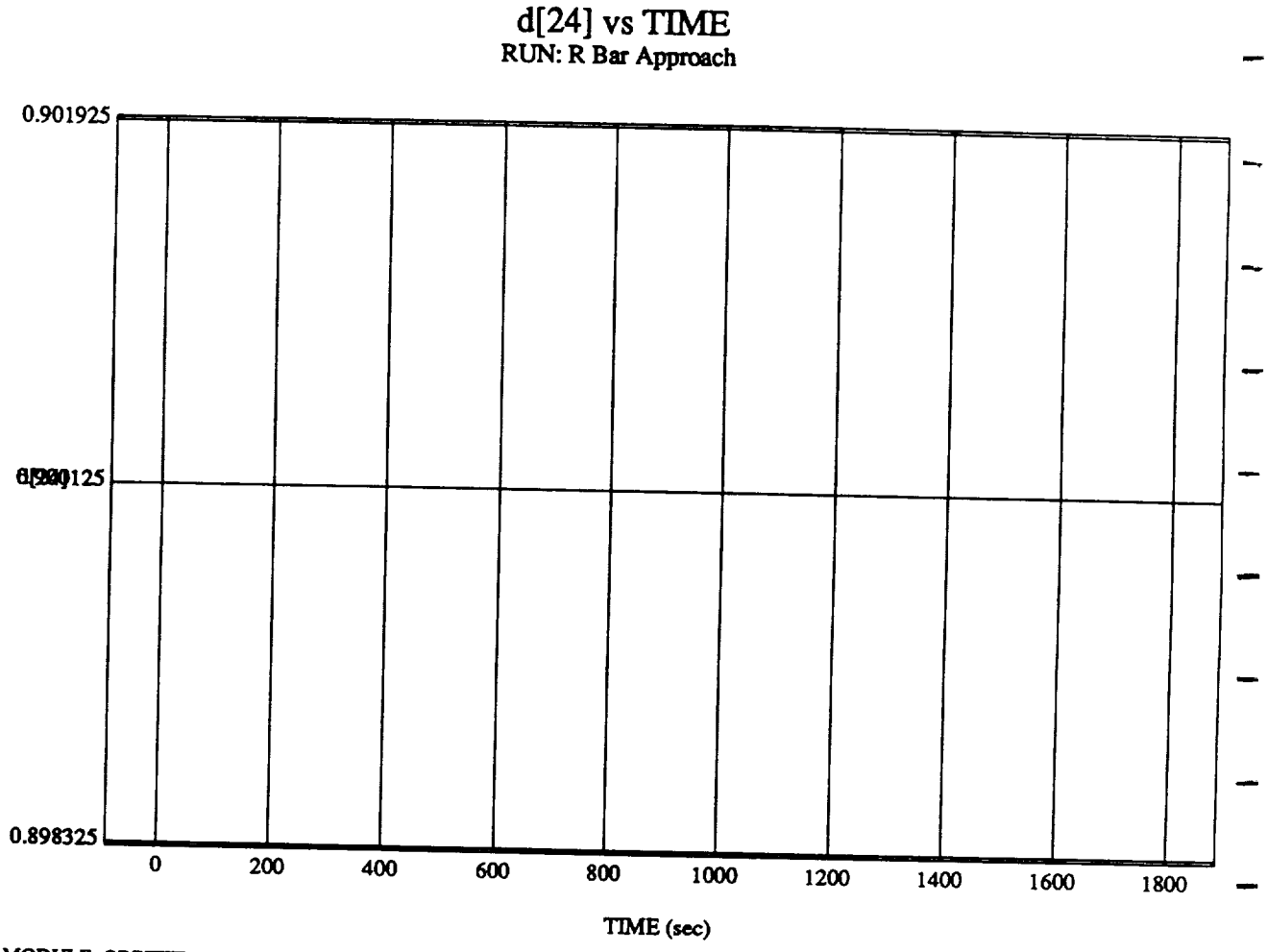
MODULE: ORBITER.lm_range
DATA SAMPLING FREQUENCY: 0.200 Hz



MODULE: ORBITER.lm_range
DATA SAMPLING FREQUENCY: 0.200 Hz



MODULE: ORBITER.lm_range
DATA SAMPLING FREQUENCY: 0.200 Hz



MODULE: ORBITER.lm_range
DATA SAMPLING FREQUENCY: 0.200 Hz

RICIS Preface

This research was conducted under auspices of the Research Institute for Computing and Information Systems by Dr. Yashvant Jani of the Technology Systems Division of Togai InfraLogic, Inc. Dr. Kwok-bun Yue served as the RICIS research coordinator.

Funding was provided by the Information Systems Directorate, NASA/JSC through Cooperative Agreement NCC 9-16 between the NASA Johnson Space Center and the University of Houston-Clear Lake. The NASA research coordinator for this activity was Dr. Robert N. Lea of the Information Technology Division, NASA/JSC.

Research Activity AR.06

APPLICATION OF FUZZY LOGIC-NEURAL NETWORK BASED REINFORCEMENT LEARNING TO PROXIMITY AND DOCKING OPERATIONS

Deliverable D3
Report on the Shuttle Translational Control Results

submitted
to

The Research Institute for Computing and Information Systems
University of Houston-Clear Lake
Houston, Texas 77058-1096

Prepared by

Yashvant Jani
Technology Systems Division
Togai InfraLogic Inc.
Houston, Texas 77058

November 15, 1992

APPLICATION OF FUZZY LOGIC-NEURAL NETWORK BASED REINFORCEMENT LEARNING TO PROXIMITY AND DOCKING OPERATIONS

Deliverable D3
Report on Shuttle Translational Control Results

submitted
to

The Research Institute
for
Computing and Information Systems

Prepared by

Technology Systems Division
Togai InfraLogic Inc.
Houston, Texas 77058

November 15, 1992

Table of Contents

List of Figures

List of Tables

1.0 Introduction :

As part of the RICIS project # AR.06 activity, the reinforcement learning techniques developed at Ames Research Center are being applied to proximity and docking operations using the Shuttle and Solar Maximum Mission (SMM) satellite simulation. In utilizing these fuzzy learning techniques, we also use the Approximate Reasoning based Intelligent Control (ARIC) architecture, and so we use two terms interchangeable to imply the same. This activity is carried out in the Software Technology Laboratory utilizing the Orbital Operations Simulator (OOS).

This report is the deliverable D3 in our project activity and provides the test results of the fuzzy learning translational controller. This report is organized in six sections. Based on our experience and analysis with the attitude controller, we have modified the basic configuration of the reinforcement learning algorithm in ARIC as described in section 2. The shuttle translational controller and its implementation in fuzzy learning architecture is described in section 3. Two test cases that we have performed are described in section 4. Our results and conclusions are discussed in section 5, and section 6 provides future plans and summary for the project.

2.0 Fuzzy Learning System Configuration

In this section, we have described the changes/modifications we have made in the fuzzy learning algorithms within the ARIC framework. These changes were determined necessary based on our study so far to properly utilize the fuzzy learning techniques for space operations. The ARIC algorithm with these changes provides us a baseline that we will use for all translational modules - range, azimuth and elevation control.

a. Remove bias from all inputs and rules.

We have learned from the attitude controller performance that the bias term really limits the fuzzy controller's ability to perform, especially when the bias term is kept constant at 0.5 value through out the learning cycle. The bias was zeroed out in several test cases and results showed good improvements, particularly in learning the environment. Therefore, we have decided to remove bias from our future implementations. We have also discussed this issue with Dr. Berenji at Ames and he also agrees that if we do not need bias, then, it would be better for the learning system.

b. Updates of d's and f's both depend on the belief value of premise as well as consequent part of the rule.

In our earlier experiments, we observed that the variations in all d's and all f's were the same, meaning that if d(1) changes by an amount, then d(2) will change by the same amount, even though, the initial values of d(1) and d(2) are different. This behavior is not acceptable in the sense that the coefficients d's and f's should have different variations based on which rules fire. We examined the formulas that update these weights, and concluded that the best way to incorporate effects of rule firing is to include the strength in the update of coefficient f. Similarly, the d's should be updated using the belief value from the fuzzification of the premise side of the rule. Both of these changes were discussed in detail with Dr. Berenji, Dr. Lea and our team. It is very appropriate to utilize these changes in the algorithm, and therefore, we implemented these modifications.

c. Crisp failure with protection.

Our current implementation uses what we call crisp failure, e.g. 0 or -1. at 1.4 DB, but we must protect the weight updating from too much punishing. Since failure is based on a parameter value, it can continue to be a failure for many cycles. If the weights are updated for few cycles with a large punishment, the controller goes unstable and can not recover at any time during the mission. This is not allowed and will not be allowed for space operations. So we need to protect against too much punishing that can result in total catastrophe.

d. Action changed to no action rather than reverse action.

In the earlier version of ARIC, the Stochastic Action Modifier (SAM) changed the 'push' from one way to the other way, if the random probability exceeded the measure of confidence p calculated using the Action Selection Network coefficients. This means that the controller requires a positive torque, but the random probability at that time is larger than the measure of confidence p (output of Action Selection network), then SAM changed the positive torque to negative torque. { 'push' is changed to '-push' }. This process for space operations is not acceptable because the negative torque in such case will perturb the state too much and the controller may not be able to recover from such an action.

In space operations, typically, an action is taken or no action is taken. It is seldom the case that an action is changed to its reverse. We also understand that such a process is desired in learning process, because the fuzzy learning technique is exploring all possible solution space. However, if we want to implement these learning techniques on a spacecraft so that it learns real time, this process must be changed to lower the risks. Therefore, we suggest to use the following procedure.

If the random probability is larger than the measure of confidence p, then SAM should change the action to no action, meaning that the push value is set to zero, and not to its negative. We accomplish several benefits from this type of process.

First, the controller will still learn from the search of solution. Now it is learning if the action recommended by the controller is really working or not. It really learns what happens when the action is not taken. Second, the controller will not be punished too much and thus it will be possible to maintain the performance rather than degrade it. Third, the solution will be slowly discarded rather than catastrophic failure. The weight updates during this time may result in a better solution. Fourth, process like this one is acceptable for space operations without increasing the risk of failure. Operationally, it is never acceptable to change an action to its reverse until there is a very high confidence that the reverse action is really needed. In our technique, since we have low confidence in an action, it should not be interpreted as high confidence in reverse action. Thus, logically it also makes sense for caring out operations.

e. No changes to computations for measure of confidence.

The measure of confidence 'p' is computed as originally proposed rather than normalizing using the number of rules. During the analysis of attitude controller performance we suggested that the measure of confidence p should be normalized per rule so that it will not continue to saturate during the learning process. We performed several attitude controller tests with this normalization, and had observed good learning rate for the neural network. However, now our main concern stems from the fact that the decision to fire jets was not arrived at by firing just one rule. Entire rulebase was utilized in the process. so for that reason we should not normalize the p-value. One can also argue that all rules were not used in deriving the controller action. We should properly scale the p-value using the number of rules fired. If we know that only four rules are responsible for the decision, then, we can normalize using these four rules. However,

it is not possible to find out which four rules are responsible. Thus, it is best not to normalize the measure of confidence. Furthermore, the d's and f's are being updated using the firing strength from the left hand side and right hand side of the rules anyway. Thus, the calculation of 'p' will include effect of long term behavior.

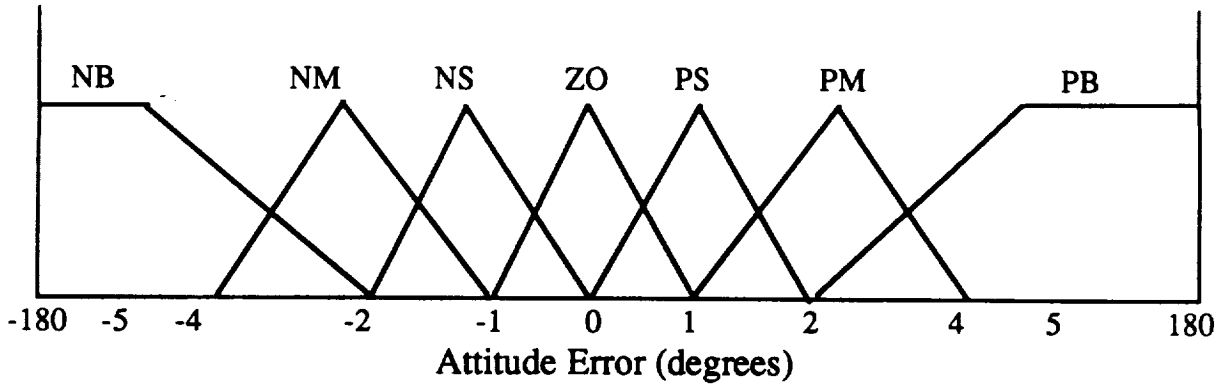
f. Input parameters normalized between 0.0 and 1.0.

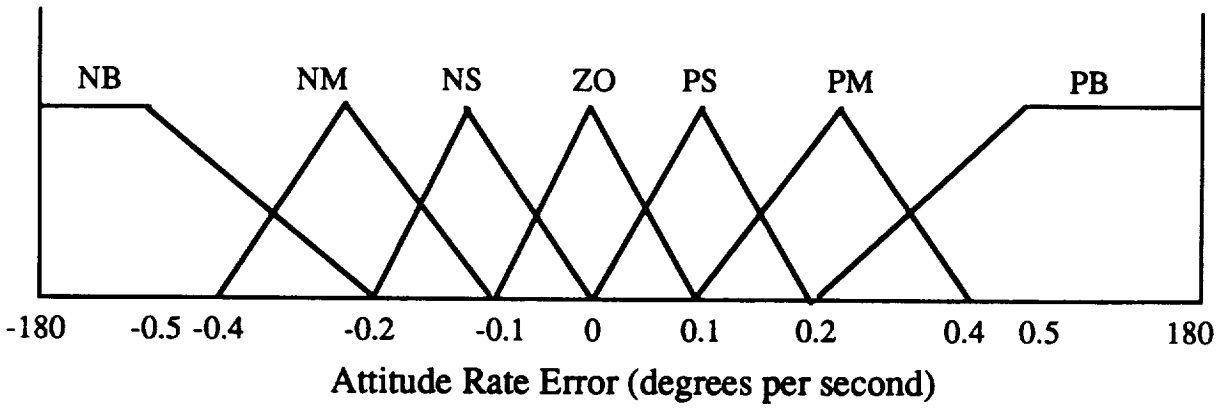
We were first normalizing the input parameters phi error and phi_dot error between -1.0 and +1.0. The reinforcement learning algorithms at this time require that the input parameters be normalized between 0.0 and +1.0. We have changed the attitude controller to reflect this change. In our opinion, there are no guidelines in neural network algorithms how to normalize input and output parameters. It is a research area to generate guidelines which parameters should scaled between 0.0 and 1.0, and which one should be scaled between -1.0 and +1.0. For example, range has no negative values, and there is what so ever no interpretation when a sensor measure negative range except that the sensor is failed. Similarly, mass of an object is never negative.

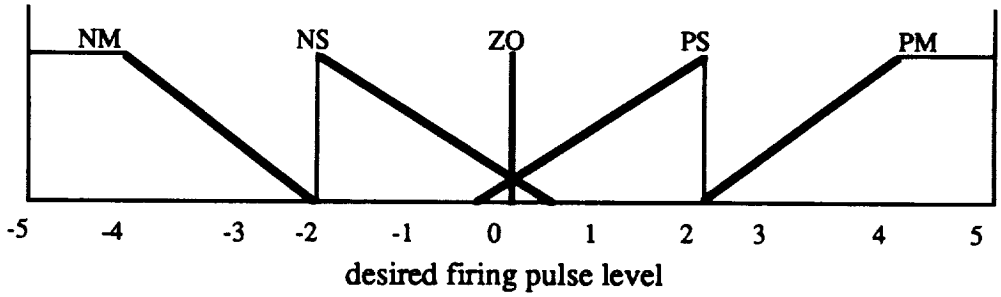
g. Overlapping Membership Functions.

During the analysis of attitude control performance, we discovered that the fuzzy learning technique uses a lot more fuel compared to fuzzy controller alone. Our results showed that the fuzzy learning or ARIC used three to eight times the fuel used by fuzzy only control. Further analysis showed that the defuzzification by Tsukomoto's method does not allow to use triangular membership functions for output. As a result we had to use only one side of the triangle. When we used only one side of the triangle, it resulted in a large hysterisis in the jet firing command sequence. Once the jets are turned on to provide torque in a desired direction, they are not turned off as soon as the rate error is corrected. The jets remain on for additional three to four cycles providing increased rate in the reverse direction. The angle error then hits the other side of the deadband. The control system then fires jets to reduce the angle error that was really caused by the previous unnecessary jet firing. Net effect is larger fuel usage and more angular activity.

We further analyzed the rate error membership functions, and decided to overlap them (as shown in Fig. 1) to reduce the net hysterisis. We performed several tests to find a proper overlap, and now we have reduced the hysterisis to no hysterisis. We have decided to use this attitude controller in all our future tests. The rulebase for the attitude controller is shown in Table I for completeness. The translational controller uses this new attitude controller with overlap in rate membership functions. However, we will not utilize such overlap in rate errors for translational controller because we need to analyze the performance without such overlap. At this time we do not know if a hysterisis exists in the translational control.







KEY:
NB - Negative Big, NM - Negative Medium, NS - Negative Small, ZO - Zero,
PS - Positive Small, PM - Positive Medium, PB - Positive Big

Fig. 1 Attitude Controller Membership functions -
Overlapping Rate Error MBF's

Table I. Fuzzy Rulebase for Attitude control

		Angle Error						
		NB	NM	NS	ZO	PS	PM	PB
Rate Error	NB	PM	PM	PS	PM			
	NM	PM	PM	PS	PM			
	NS	PS	PS	PS	PS			
	ZO	PS	PS	PS	ZO	NS	NS	NS
	PS				NS	NS	NS	NS
	PM				NM	NS	NM	NM
	PB				NM	NS	NM	NM

KEY:
NB - Negative Big, NM - Negative Medium, NS - Negative Small, ZO - Zero,
PS - Positive Small, PM - Positive Medium, PB - Positive Big

3.0 Fuzzy Logic based Shuttle Translational Controller

We have implemented the shuttle translational controller using TILShell and fuzzy-C compiler in the software technology laboratory. Development of this translational controller and its performance was reported at the AIAA Guidance, Navigation, and Control Conference in 1991. We plan to utilize the same controller with minimum changes in the ARIC architecture.

The control tasks during the proximity operations are : 1) maintain desired range by controlling the thrust along the line of sight, 2) keep the elevation angle close to zero with respect to line of sight, and 3) keep the azimuth angle close to zero with respect to line of sight. It is expected that the attitude controller is performing with desired accuracy in maintaining the desired angle and rate. This assumption is necessary because the translational controller is slaved after the attitude controller in the shuttle manual operations.

The input parameters to the translational controller are range, range rate, elevation angle, azimuth angle, elevation rate, and azimuth rate as shown in Fig. 2. In reality only three parameters are measured and their rates are derived using differencing method at regular intervals. The output parameters are the translational hand controller commands (Fig. 2) that fire the jets in a given direction. There are three hand controller commands known as THC-X, THC-Y, and THC-Z used in jet select logic to fire the jets for thrust in x, y and z directions with respect to the shuttle. Typically range and range rates are used to generate the approach thrust, and the elevation and azimuth angles are used to generate side thrusts to maintain the line of sight. The THC axes are transformed into appropriate direction by transformation matrix that involves the attitude of the shuttle in the local horizontal local vertical coordinate frame. The software for this transformation is already a part of the flight software in OOS.

We have designed the membership functions for all input and output parameters using the baseline earlier developed. Membership functions for elevation error, elevation rate error and its corresponding commanded delta-v are shown in Fig. 3, and its rulebase is shown in Table II. Since the elevation error membership functions are designed specially for the Shuttle operations, they are very asymmetric with respect to the positive and negative values in their Universe of Discourse. This asymmetry is required because the shuttle's center of mass is very far away from the radar location from where the parameters are measured. Since the ARIC architecture can not use triangular membership functions in the output set, we have used only one side of the triangle as shown in Fig. 3. This is very similar to the attitude control output membership functions shown in Fig. 1. Membership functions for azimuth control are shown in Fig. 4, where the azimuth error and its rate error are input and the commanded delta-v is output. The rulebase is shown in Table III. The range error, range rate error and commanded delta-v membership functions for the relative distance control are shown in Fig. 5, and the rulebase that generates these commanded delta-v's using range error and range rate error as two inputs is shown in Table IV. Please note that the output parameter membership functions are one sided triangles because the way ARIC defuzzification method is implemented.

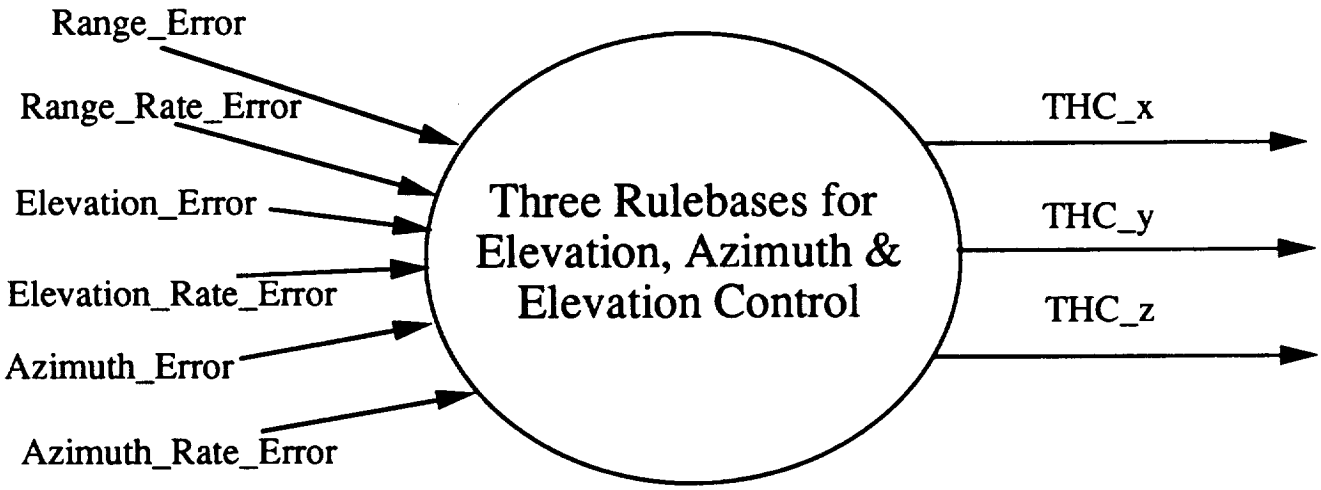
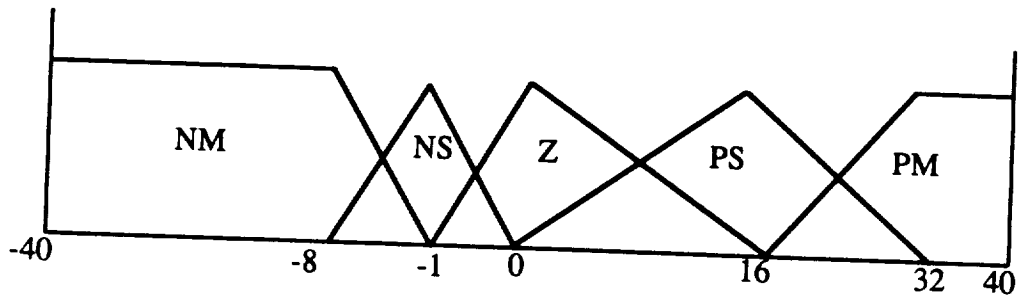
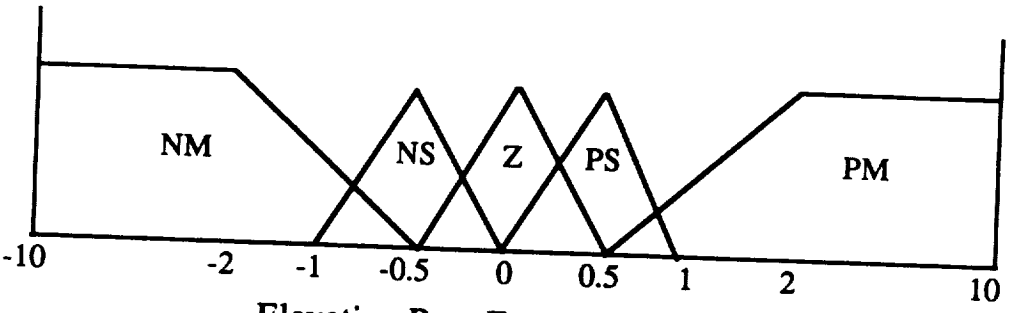


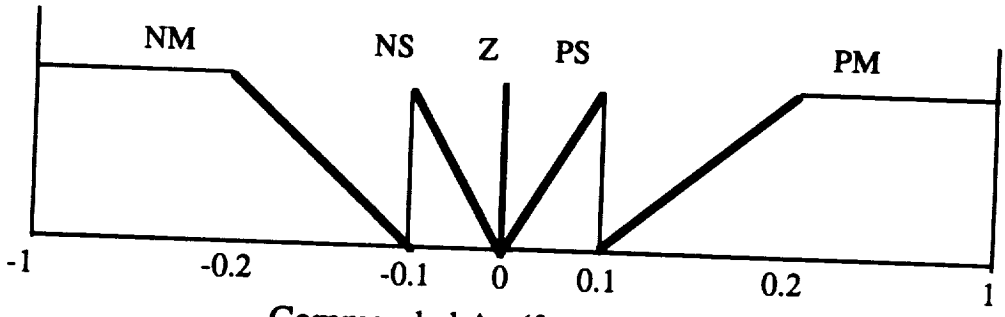
Fig. 2 Definition of Relative Trajectory Control



Elevation Angle Error (degrees)



Elevation Rate Error (deg. per sec.)



Commanded Δv (feet per sec.)

Fig. 3 Membership Functions for Elevation Control

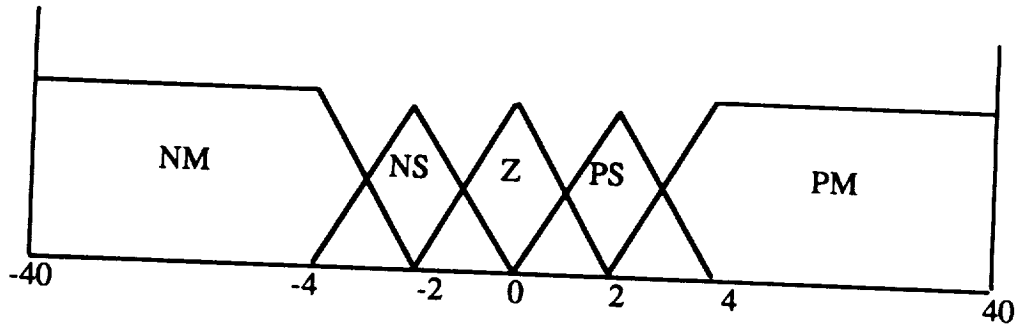
Table III. Rulebase for azimuth control

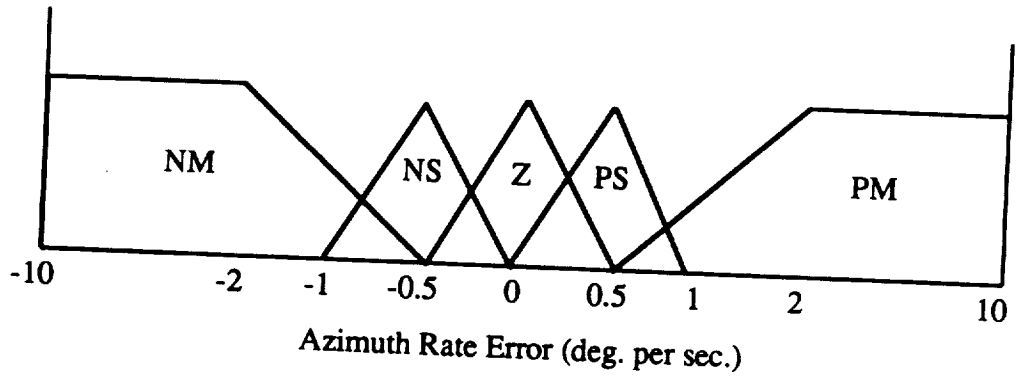
		AZIMUTH RATE ERROR				
		NM	NS	ZO	PS	PM
	PM			NM	NM	NM
AZIMUTH	PS			NS	NS	NM
ANGLE	ZO	PM	PS	ZO	NS	NM
ERROR	NS	PM	PS	PS		
	NM	PM	PM	PM		

Table II. Rulebase for elevation control

		ELEVATION RATE ERROR				
		NM	NS	ZO	PS	PM
	PM			NM	NM	NM
ELEV.	PS		ZO	NS	NS	NM
ANGLE	ZO	PM	PS	ZO	NS	NS
ERROR	NS	PM	PS	PS	ZO	
	NM	PM	PM	PM		

KEY:
NB - Negative Big, NM - Negative Medium, NS - Negative Small, ZO - Zero,
PS - Positive Small, PM - Positive Medium, PB - Positive Big





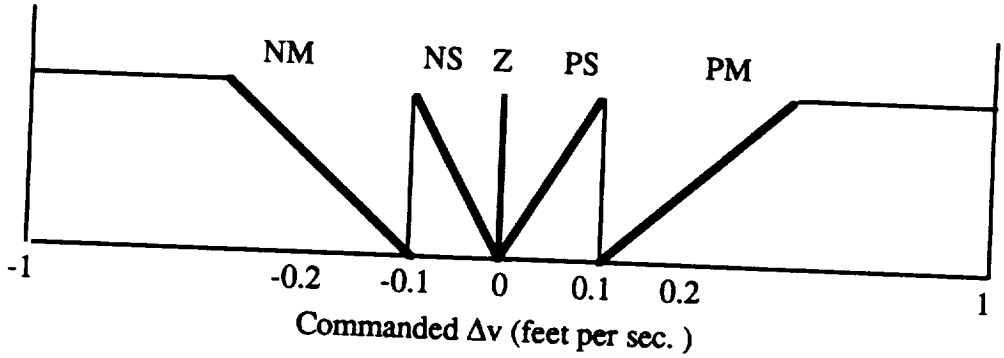
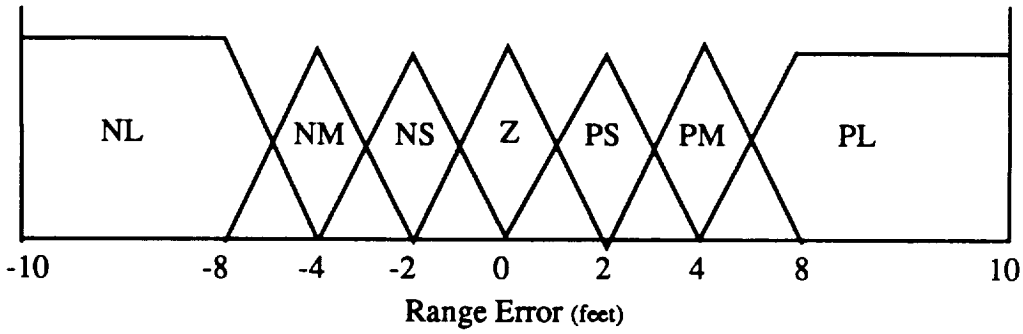
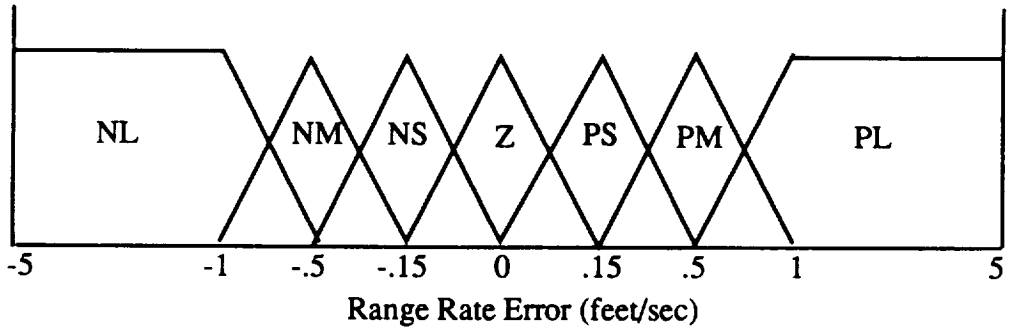


Fig. 4 Membership Functions for Azimuth Control





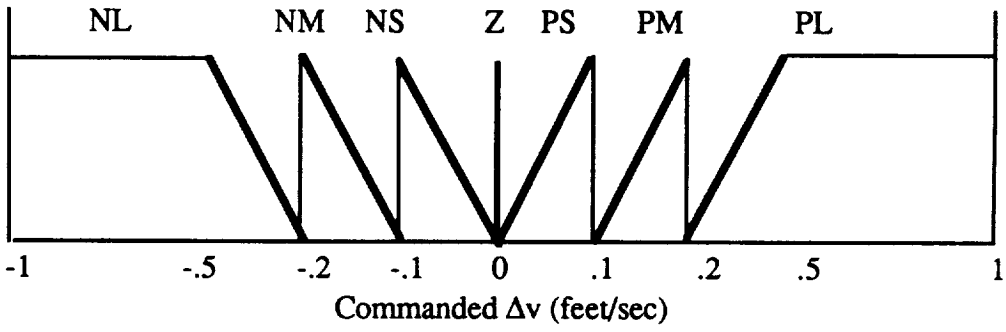


Fig. 5 Membership Functions for Range Control

Table IV. Rulebase for range control

		RANGE RATE ERROR						
		NB	NM	NS	ZO	PS	PM	PB
RANGE ERROR	PB					NB	NB	NB
	PM					NM	NB	NB
	PS					NS	NM	NB
	ZO	PM	PS	PS	ZO	NS	NS	NM
	NS	PB	PM	PS				
	NM	PB	PB	PM				
	NB	PB	PB	PB				

KEY:
NB - Negative Big, NM - Negative Medium, NS - Negative Small, ZO - Zero, PS - Positive Small, PM - Positive Medium, PB - Positive Big

4.0 Description of Test Cases

For our translational controller we have performed four testcases. These testcases were also performed for the fuzzy logic based 6 DOF controller and we have the man-in-the-loop simulation data that provides us insight as to how the Shuttle crew is trained for performing these proximity operations.

The first test case is the v-bar approach where the shuttle is initialized at 400 feet distance in the front of the satellite. The pitch attitude of the Shuttle is 90 degrees (as shown in Fig. 6) so that a crew member can see the satellite from the window or the Crew Optical Alignment System (COAS) for measuring the angles of the satellite relative to the line of sight. In our case we are using a radar like sensor placed at its proper location for measurements. The Shuttle has a closing rate of 0.4 feet per second. Its mission time-line calls for maintaining certain closing rate as a function of distance. As it approaches 50 feet distance, the mode changes to Station-keeping during which this distance is maintained. It approximately takes 1400 seconds for the Shuttle to complete this transition from 400 feet to 50 feet. The second test case known as r-bar approach is similar to the first one. Instead of v-bar it approaches along r-bar. The Shuttle is initialized at 400 feet on r-bar which is the z-axis of the LVLH frame as shown in Fig. 6. The pitch attitude of the Shuttle is zero degrees, and it sees the satellite from the window or COAS or radar. The trajectory is as shown in Fig. 6.

The third test case is the Fly-Around test case. The Shuttle is initialized at 200 feet on v-bar with pitch attitude of 90 degrees very similar to the v-bar case. However, its pitch rate is initialized at 0.05 deg per sec, and thus it will rotate to increase the pitch angle. As the Shuttle rotates, the elevation angle will change and the range rate may also change. The elevation control rulebase will see this change in the elevation angle and fire the jets to move the Shuttle upward. As a results, the Shuttle will translate upward of v-bar. As the pitch rate continues to be non-zero, the elevation angle will continue to change, and the elevation control will keep the Shuttle pushing upward. When the Shuttle moves sufficiently upward, its range will increase, and then, the range control will initiate the range rate to reduce the range to 200 feet. Thus, the Shuttle will traverse a circular path as shown in Fig. 7, and will reach the negative r-bar in a certain given time. Actually, the initial pitch rate of 0.05 will set the arrival time of 1800 seconds. If the 90 Fly-around is desired in 900 seconds, then, the initial pitch rate should be 0.1 deg per sec. The mission time-line is typically set using this type of calculations.

The fourth testcase is the station-keeping testcase, where the Shuttle is initialized at 200 feet distance on v-bar with 90 deg pitch angle and zero pitch rate. The range must be maintained within the range deadband as the mission time increases. Also, the elevation and azimuth angles must be maintained near zero. This activity is very simple and very boring for the Shuttle crew. There is not much jet firing activity and the drift in range is very slow, so the Shuttle is not moving relatively fast. However, this station-keeping is an important activity because it allows the Shuttle crew to check out sensors and other systems. We have set up this activity for 1800 seconds to be consistent with all other testcases.

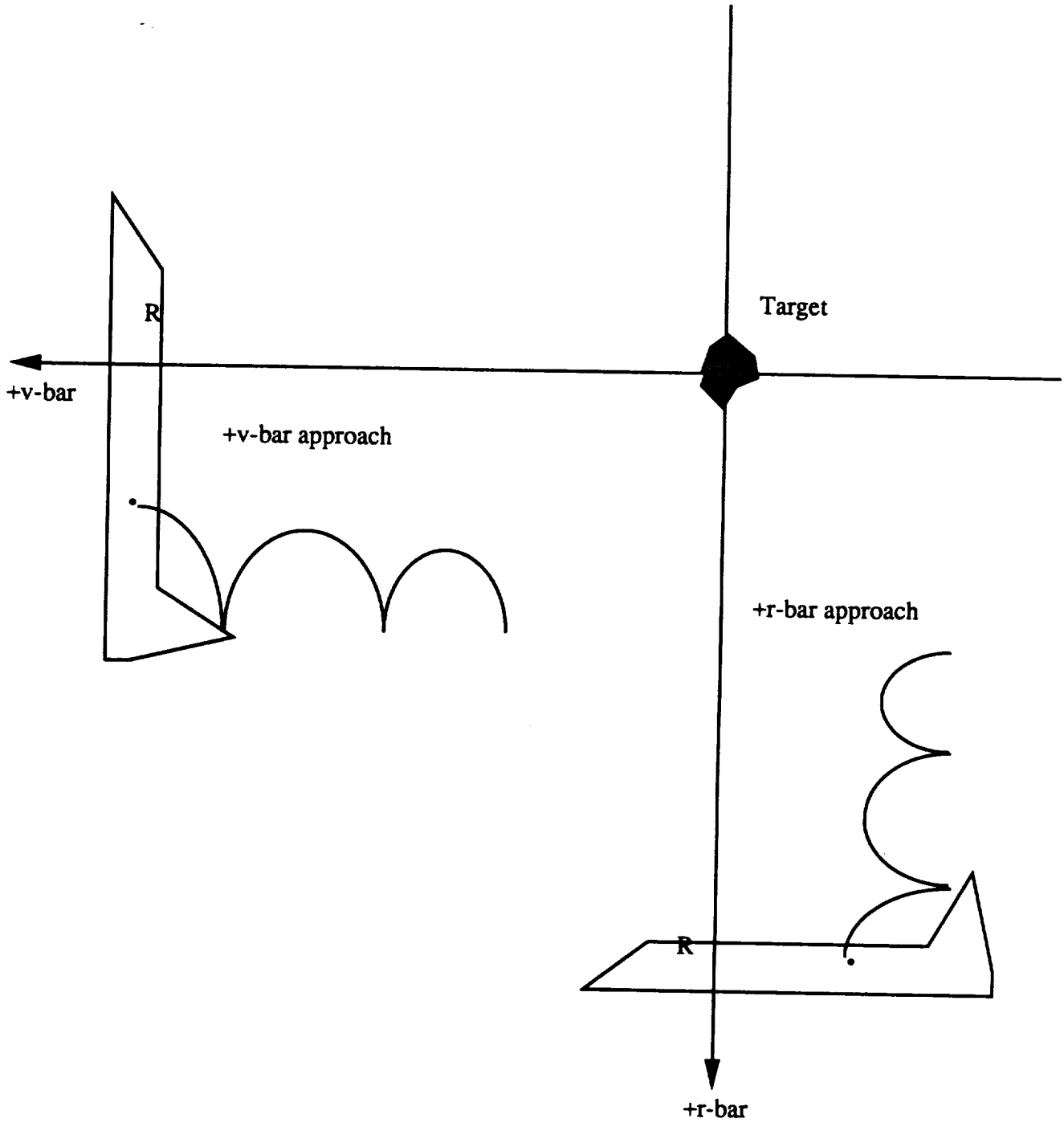


Fig. 6 Definition of Proximity Operations v-bar and r-bar Approaches

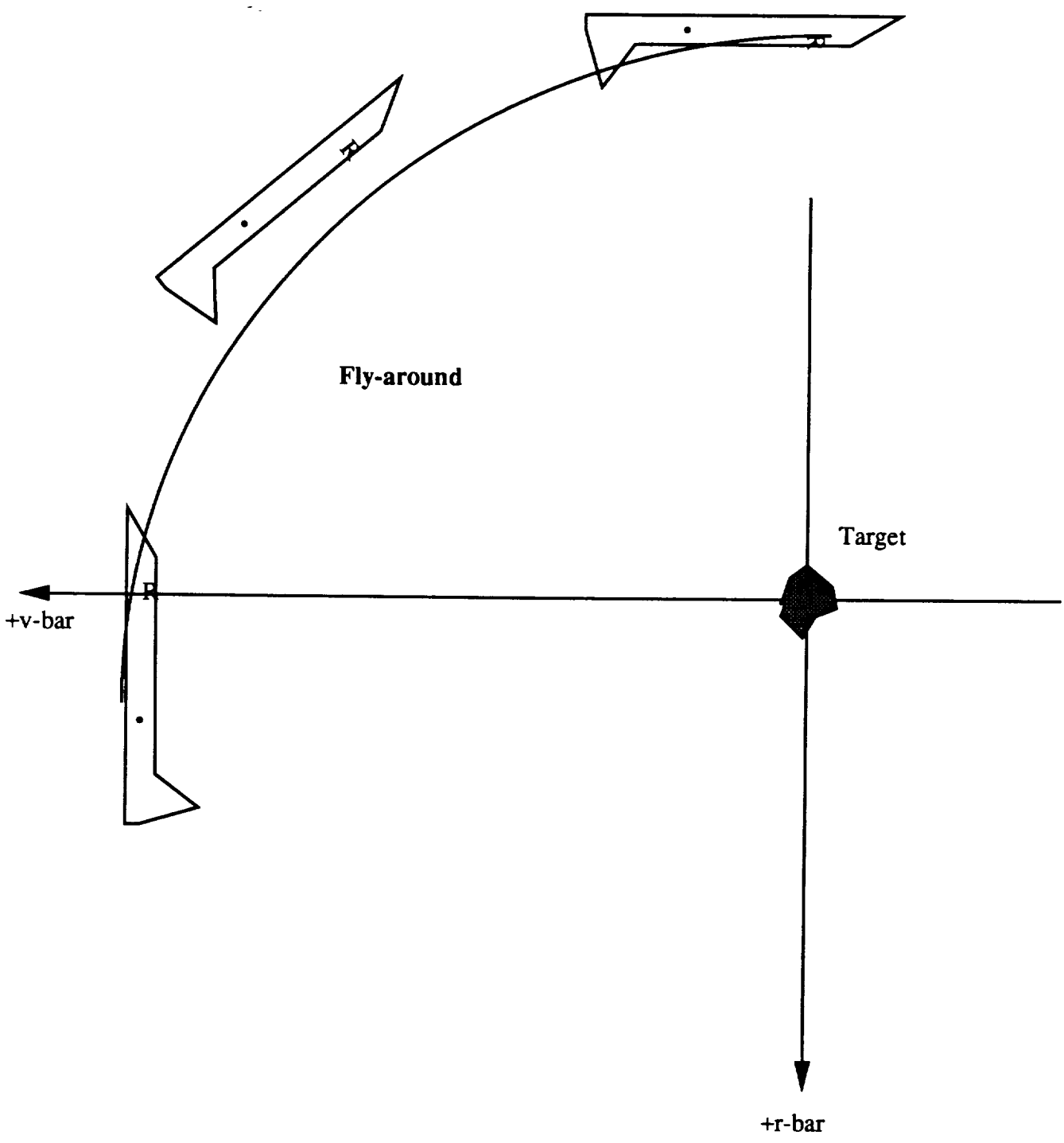


Fig. 7 Definition of Proximity Operations fly-around Segment

5.0 Results and Conclusions

In all four testcases, the translational controller in ARIC architecture works ; it achieves the desired position at the end of the proximity operations segment. This behavior is expected because the translational controller derived in ARIC is from the fuzzy logic based controller which has given very good results. In all testcases, we observed that the fuel usage is comparatively large. Since we have not tried to optimize the membership functions, particularly the rate membership functions, to achieve any fuel efficiency or any performance improvement, this behavior is again expected for this controller.

The Action Evaluation Network and Action Selection Networks do not learn much because of short mission duration time. Each of the testcase is only 1800 seconds long. In our attitude controller testcase, these two networks learned the environment over 50,000 seconds training time. Coefficients d's and f's were stabilized after this long training. We suggest that we use the station-keeping testcase for 100,000 seconds to train these two networks and see what happens. Any other testcase is not feasible to use because it will require to set up initial conditions again and again.

Coefficients d's and f's are being updated properly and the networks seem to handle these updates favorably. The Fly-Around trajectory is not that good in comparison to what an experienced crew member will fly. However, this is expected because the networks are in training. Trajectories in other testcases seem to be expectable. The scaling of input parameters is based on full Universe of Discourse resulting in a small variation of inputs within 0.0 and 1.0 range. Typically, we expect that x(0) and x(1) should vary between 0.2 and 0.8 for proper learning. In current testcases, it varies between 0.45 and 0.55 or even less. This is really not good for network training.

6.0 Future Plans and Summary

First of all we plan to change the scaling properly so that the input parameter variations are within 0.2 to 0.8 range. To achieve such scaling, we will have to change the source code of all three modules and re-compile them. We will perform all testcases again and generate all plots for detail analysis. Next we plan to overlap the rate membership functions and try to optimize the fuel usage. Based on the results, we expect that there is a hysterisis within the rate membership functions and we should remove it by modifying the rate membership functions as we did for the attitude controller.

We then plan to set up a test case that will simulate the shuttle docking operations. In this test case, the shuttle will approach the solar max satellite from 50 feet to 2 feet and hold the relative orientation for a specified time at the final distance so that the grappling task can be performed. We will analyze the performance of ARIC and learning modules for this test case and provide a presentation and proper documentation for the results.

A1. Source code for range control

```
#include <stdio.h>
#include <math.h>
#include <sys/types.h>

        /* EXTERNAL DATA STRUCTURE DEFINITION */
#include "../orb_fuzzy/learn_cycle.h"
#define max(x,y)      ( (x >= y) ? x : y )
#define min(x,y)      ( (x < y) ? x : y )
#define Gamma         0.9
#define Beta          0.2
#define Beta_h        0.05
#define Rho           1.0
/* 1 July 92 Change Rho_h from 0.2 to 0.8 */
#define Rho_h         0.8
#define Rho1          2.0
#define Rho_h1        0.4
extern double sgn();
extern double exp();
extern double rnd();
extern double sim_time();

learn_range(L)
LEARN_CYCLE * L;        /*IN : */
{
  int i,i1, j, k;
  double range_match(), range_calculate_z_array();

  double temp ;


       L->x[0] = (L->Phi+10)/20 ;
       L->x[1] = (L->Phi_dot + 5)/10.0 ;

     /* CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC */
     /* 1 July 1992 - Set Bias to 0.0              */
     /* CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC */
     L->x[2] = 0.0 ;

     L->failure = 0;

     /* Set up and evaluate the failure criteria */
        if ( (fabs(L->Phi) > 10) || (fabs(L->Phi_dot) > 5) ) {
          L->failure = -1.;
          if( L->learn_flag ) {
            fprintf(stderr,"learn_range: Failure at %f.\n",sim_time());
          } /* end if */
        } /* end if */

   /* CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC */
   /* CC 6 November, 1992                              CC */
   /* CC Turn off learning when there is a failure     CC */
   /* CC until there has been no failure for 4 pass    CC */
   /* CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC */

   L->fourth_pass_failure = L->third_pass_failure ;
   L->third_pass_failure = L->second_pass_failure ;
   L->second_pass_failure = L->first_pass_failure ;
   L->first_pass_failure = 0 ;
   if(L->failure == -1) {
     L->first_pass_failure = 1 ;
   } /* end if */

   /* If No Failure in the last 4 passes then turn learning on */
   if( !L->fourth_pass_failure &&
```

```
    !L->third_pass_failure  &&
    !L->second_pass_failure  &&
    !L->first_pass_failure ) {
  L->learn_flag = 1 ;
} /* end if */

/* If Failure first pass, but not second pass, let the learn
  flag stay on, so that the failure is processed. On the next
  pass (where there is a failure second pass, but not third
  pass) then turn learning off until there are no failures for
  four passes */
if( L->second_pass_failure && !(L->third_pass_failure) ) {
  L->learn_flag = 0 ;
} /* end if */


/* output: state evaluation */

for (i = 0; i < 25; i++)
  {
    L->sum = 0.0;
    for(j = 0; j < 3; j++)
      {
        L->sum += L->a[i*3+j] * L->x_old[j];
      }
    L->y[i] = 1.0 / (1.0 + exp(-1.0 * L->sum));
  }
L->sum = 0.0;
for(i = 0; i < 25; i++)
  {
    L->sum += L->c[i] * L->y[i];
  }
L->sum1 = 0.0;
for ( j = 0;j < 3; j++)
  {
    L->sum1 += L->b[j] * L->x_old[j];
  }
L->v = L->sum + L->sum1;

/* output: action */
for(i = 0; i < 25; i++)
  {
      i1=i;

    L->w[i] = range_match(i1,L);
    L->z1[i] = range_calculate_z_array(i1,L);
  }
L->num1 = 0.0;
L->denom = 0.0;
for(i = 0; i < 25; i++)
  {
    L->num1 += L->w[i] * L->z1[i] * L->f[i] ;
    L->denom += L->w[i]*L->f[i] ;

  }

/* JUNE 9, 1992 - CORRECTION !!! */
/* Add test for denom very small compared to sum1 - no rule firing zone */
/* L->push = (1000.0*fabs(L->denom)<fabs(L->sum1))?0.0:L->sum1/L->denom;*/
if(fabs(L->num1)<=0.01 || fabs(L->denom)<=0.01) {
  L->push = 0.0 ;
} else {
  L->push = L->num1/L->denom ;
} /* end if */
```

```
    /* output: action computations completed */
    for(i = 0; i < 25; i++)
      {

        L->sum = 0.0;

        for (j = 0; j < 3; j++)
          L->sum += L->d[i*3+j] * L->x_old[j];

          L->z[i] = 1.0 / (1.0 + exp(-1.0 * L->sum));
      }
    L->sum2 = 0.0;
    L->sum3 = 0.0;

    for(i = 0; i < 3; i++)
      L->sum2 += L->e[i] * L->x_old[i];

    for (i=0;i < 25; i++)
      L->sum3 += L->f[i] * L->z[i];

/*CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CC JULY 1, 1992 - Change Normalize of sum4                          CC
CC L->sum4 = L->sum3 + L->sum2;                                     CC
CC JULY 1, 1992 - Normalize of sum3 by # rules                      CC
CC L->sum4 = L->sum3 / 31.0 + L->sum2 / 3.0 ;                       CC
CC OCT 2, 1992 - Do NOT normalize for # rules                       CC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC*/

L->sum4 = L->sum3 + L->sum2 / 3.0 ;

L->p = 1.0 / (1.0 + exp(-1.0 * L->sum4));

    /* CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC */
    /* 15 April 1992 - Use temp variable - not push                    */
    /* CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC */
    /* L->push = (rnd() <= L->p) ? L->push : -L->push;                 */
    /* L->unusualness = (L->push > 0) ? 1.0 - L->p : -L->p; */

    /* CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC */
    /* 28 Sept 1992 - Set stocastic action to zero out cmd  */
    /* CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC */
    /* L->push = ( rnd() <= ((L->p+1.0)/2.0) ) ? L->push : -L->push; */

/* CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
   28 Sept 1992 - Change definition of unusualness to
   correspond with Hamid's notes.
L->push = ( rnd() <= ((L->p+1.0)/2.0) ) ? L->push : 0.0 ;
L->unusualness = (L->push > 0) ? 1.0 - L->p : -L->p;
*/
    if( rnd() <= ((L->p+1.0)/2.0) ) {
      L->unusualness = 1.0 - L->p ;
    } else {
      L->push = 0.0 ;
      L->unusualness = -L->p ;
    } /* end if */


    /*   using new input values and unmodified weights. */
    /*   Use y_new and v_new so not to destroy y and v. */
    for(i = 0; i < 31; i++)
      {
        L->sum = 0.0;
        for(j = 0; j < 3; j++)
```

```
          {
            L->sum += L->a[i*3+j] * L->x[j];
          }
        L->y_new[i] = 1.0 / (1.0 + exp(-1.0 * L->sum));
      }
        L->sum = 0.0;
        L->sum1 = 0.0;
        L->sum2 = 0.0;

    for(j = 0; j < 3; j++)
        L->sum1 += L->b[j] * L->x[j];

    for(i = 0; i < 25; i++)
        L->sum2 += L->c[i] * L->y_new[i];

    L->sum = L->sum1 + L->sum2;
    L->v_new = L->sum;


    /*CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC*/
    /* 17 September, 1992                                                */
    /* This logic depends upon a "crisp" (two valued) failure  */
    /* It has been replaced by "fuzzy" failures                          */
    /* action evaluation */
        if (L->failure)
          L->r_hat = L->failure - L->v;
        else
          L->r_hat = L->failure + Gamma * L->v_new - L->v;
    /*CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC*/
/*  L->r_hat = -1.0 * L->failure +
/*              ( 1.0 - L->failure ) * Gamma * L->v_new - L->v ;

    /* modification and update to parameters */

        for(i = 0; i < 25; i++)
          {
            L->factor1 = Beta_h * L->r_hat * L->y[i] * (1.0 - L->y[i]) * sgn(L->c[i]);
            L->c[i] += Beta * L->r_hat * L->y[i];

             for(j = 0; j < 3; j++)
                  {
                    L->a[i*3+j] += L->factor1 * L->x_old[j];
                  }
          }


        for(i = 0; i < 3; i++)
                L->b[i] += Beta * L->r_hat * L->x_old[i];

        for(i = 0; i < 25; i++)
          {
/*CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CC 7 July, 1992 : Weight updates of D's by rule firing strength CC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC*/
            L->factor2 = Rho_h * L->r_hat * L->z[i] * (1.0 - L->z[i]) * sgn(L->f[i])*L->unus
ualness ;

            for(j = 0; j < 3; j++) {
              if( L->learn_flag ) {
                L->d[i*3+j] += L->factor2 * L->x_old[j] * L->d_weights[i*3+j] ;
              } /* end if */
            } /* end for */

          }
```

```
      for(i = 0; i < 25; i++)
{

/*CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CC July 2, 1992 : Include Rule Firing Weight in F update          CC
CC L->f[i] += Rho * L->r_hat * L->unusualness * L->z[i];          CC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC*/
        if( L->learn_flag )  {
           L->f[i] += Rho * L->r_hat * L->unusualness * L->z[i] * L->w[i] ;
        } /* end if */
 }
      for(i = 0; i < 3; i++)
{

        L->e[i] += Rho * L->r_hat * L->unusualness * L->x_old[i];
 }
        L->x_old[0] = L->x[0];
        L->x_old[1] = L->x[1];
        L->x_old[2] = L->x[2];
}
/*************************************************************************/

double sgn(x)
   double x;
{
  if (x < 0.0)
    return (-1.0);
  else if (x > 0.0)
    return (1.0);
  else
    return (0.0);
}


/* zero_one function returns 0 for negative numbers
                            1 for values > 1
                            x for values between 0 and 1 */

double zero_one(x)

  double x;
{
  if (x < 0)  return (0.0);
  else if (x > 1) return (1.0);
  else return (x);
 }


/*********** Membership Functions for Range (Phi) ******************************/

double range_nb1(x)
double x;
{
  return(min( max((-4-x)/4 , 0.0 ), 1.0 ));
}

double range_nm1(x)
double x;
{
  if (x <= -4.0)      return (min( max((x+8)/4 , 0.0 ), 1.0 ));
  else  return (min( max(( -x-2)/2 , 0.0 ), 1.0 ));
  }
double range_ns1(x)
```

```
  double x;
  {
    if (x <= -2.0)      return (min( max( (x+4.0)/2 , 0.0 ), 1.0 ));
     else  return (min( max( -x/2 , 0.0 ), 1.0 ));
    }
  double range_zo1(x)
  double x;
  {
    if (x <= 0)      return (min( max( (x+2)/2 , 0.0 ), 1.0 ));
    else  return (min( max( (-x+2)/2 , 0.0 ), 1.0 ));
    }
  double range_ps1(x)
  double x;
  {
    if (x <= 2)      return (min( max( x/2 , 0.0 ), 1.0 ));
    else  return (min( max( (-x+4)/2 , 0.0 ), 1.0 ));
    }
  double range_pm1(x)
  double x;
  {
    if (x <= 4)      return (min( max(( x-2)/2 , 0.0 ), 1.0 ));
    else return (min( max(( -x+8)/4 , 0.0 ), 1.0 ));
    }

  double range_pb1(x)
  double x;
  {
    return(min( max(( x-4)/4 , 0.0 ), 1.0 ));
  }
  /******************** Range-Rate Membership Functions ***************************/
  double range_nb2(x)
  double x;
  {
    return(min( max((-0.5-x)/.5 , 0.0 ), 1.0 ));
  }

  double range_nm2(x)
  double x;
  {
    if (x <= -.5)      return (min( max(( x+1.0)/.5 , 0.0 ), 1.0 ));
    else  return (min( max(( -x-.15)/.35 , 0.0 ), 1.0 ));
    }
  double range_ns2(x)
  double x;
  {
    if (x <= -.15)      return (min( max(( x+.5)/.35 , 0.0 ), 1.0 ));
    else  return (min( max(( -x/.15) , 0.0 ), 1.0 ));
    }
  double range_zo2(x)
  double x;
  {
    if (x <= 0)      return (min( max(( x+.15)/.15 , 0.0 ), 1.0 ));
    else  return (min( max(( -x+.15)/.15 , 0.0 ), 1.0 ));
    }
  double range_ps2(x)
  double x;
  {
    /* Modified from  x/.1 to put gap between ns2 and ps2 */
    /* 29 Sept 1992                                       */
    if (x <= .15)      return (min( max(x/.15 , 0.0 ), 1.0 ));
    else  return (min( max(( -x+.5)/.35 , 0.0 ), 1.0 ));
    }
  double range_pm2(x)
  double x;
```

```
{
  if (x <= .5)      return (min( max(( x-.15)/.35 , 0.0 ), 1.0 ));
  else return (min( max(( -x+1.0)/.5 , 0.0 ), 1.0 ));
  }

double range_pb2(x)
double x;
{
  return(min( max(( x-.5)/.5 , 0.0 ), 1.0 ));
}

/***************** Defuzzification Process with Delta-V Membership Functions ************/

double range_nb3(x)
double x;
{
  return(-0.2 - 0.3*x);
}

double range_nm3(x)
double x;
{
  return(-0.1 - 0.1*x);
}

double range_ns3(x)
double x;
{
  return(-0.1*x);
}

double range_zo3(x)
double x;
{
  return(0.0);
}

double range_ps3(x)
double x;
{
  return(0.1*x);
}

double range_pm3(x)
double x;
{
  return(0.1 + 0.1*x);
}

double range_pb3(x)
double x;
{
  return(0.2 + 0.3*x);
}


double range_match(i,L)
int i;
LEARN_CYCLE *L;         /*IN : */
{
double temp;
  switch (i) {

   case 0:
```

```
      L->d_weights[i*3+0] = zero_one(range_nb1(L->Phi)*L->d[i*3+0]) ;
      L->d_weights[i*3+1] = zero_one(range_zo2(L->Phi_dot)*L->d[i*3+1]) ;
      temp =min( zero_one(range_nb1(L->Phi)*L->d[i*3+0]), zero_one(range_zo2(L->Phi_dot)*L
->d[i*3+1]));
      return(temp);
   case 1:
      L->d_weights[i*3+0] = zero_one(range_nb1(L->Phi)*L->d[i*3+0]) ;
      L->d_weights[i*3+1] = zero_one(range_ns2(L->Phi_dot)*L->d[i*3+1]) ;
      temp=min( zero_one(range_nb1(L->Phi)*L->d[i*3+0]), zero_one(range_ns2(L->Phi_dot)*L
>d[i*3+1]));
      return(temp);
   case 2:
      L->d_weights[i*3+0] = zero_one(range_nb1(L->Phi)*L->d[i*3+0]) ;
      L->d_weights[i*3+1] = zero_one(range_nm2(L->Phi_dot)*L->d[i*3+1]) ;
      temp=min( zero_one(range_nb1(L->Phi)*L->d[i*3+0]), zero_one(range_nm2(L->Phi_dot)*L
>d[i*3+1]));
      return(temp);
   case 3:
      L->d_weights[i*3+0] = zero_one(range_nb1(L->Phi)*L->d[i*3+0]) ;
      L->d_weights[i*3+1] = zero_one(range_nb2(L->Phi_dot)*L->d[i*3+1]) ;
      temp=min( zero_one(range_nb1(L->Phi)*L->d[i*3+0]), zero_one(range_nb2(L->Phi_dot)*L
>d[i*3+1]));
      return(temp);
   case 4:
      L->d_weights[i*3+0] = zero_one(range_nm1(L->Phi)*L->d[i*3+0]) ;
      L->d_weights[i*3+1] = zero_one(range_zo2(L->Phi_dot)*L->d[i*3+1]) ;
      temp=min( zero_one(range_nm1(L->Phi)*L->d[i*3+0]), zero_one(range_zo2(L->Phi_dot)*
>d[i*3+1]));
      return(temp);
   case 5:
      L->d_weights[i*3+0] = zero_one(range_nm1(L->Phi)*L->d[i*3+0]) ;
      L->d_weights[i*3+1] = zero_one(range_ns2(L->Phi_dot)*L->d[i*3+1]) ;
      temp=min( zero_one(range_nm1(L->Phi)*L->d[i*3+0]), zero_one(range_ns2(L->Phi_dot)*L
>d[i*3+1]));
      return(temp);
   case 6:
      L->d_weights[i*3+0] = zero_one(range_nm1(L->Phi)*L->d[i*3+0]) ;
      L->d_weights[i*3+1] = zero_one(range_nm2(L->Phi_dot)*L->d[i*3+1]) ;
      temp=min( zero_one(range_nm1(L->Phi)*L->d[i*3+0]), zero_one(range_nm2(L->Phi_dot)*L
>d[i*3+1]));
      return(temp);
   case 7:
      L->d_weights[i*3+0] = zero_one(range_nm1(L->Phi)*L->d[i*3+0]) ;
      L->d_weights[i*3+1] = zero_one(range_nb2(L->Phi_dot)*L->d[i*3+1]) ;
      temp=min( zero_one(range_nm1(L->Phi)*L->d[i*3+0]), zero_one(range_nb2(L->Phi_dot)*L
>d[i*3+1]));
      return(temp);
   case 8:
      L->d_weights[i*3+0] = zero_one(range_ns1(L->Phi)*L->d[i*3+0]) ;
      L->d_weights[i*3+1] = zero_one(range_zo2(L->Phi_dot)*L->d[i*3+1]) ;
      temp=min( zero_one(range_ns1(L->Phi)*L->d[i*3+0]), zero_one(range_zo2(L->Phi_dot)*L
>d[i*3+1]));
      return(temp);
   case 9:
      L->d_weights[i*3+0] = zero_one(range_ns1(L->Phi)*L->d[i*3+0]) ;
      L->d_weights[i*3+1] = zero_one(range_ns2(L->Phi_dot)*L->d[i*3+1]) ;
      temp=min( zero_one(range_ns1(L->Phi)*L->d[i*3+0]), zero_one(range_ns2(L->Phi_dot)*L
>d[i*3+1]));
      return(temp);
   case 10:
      L->d_weights[i*3+0] = zero_one(range_ns1(L->Phi)*L->d[i*3+0]) ;
      L->d_weights[i*3+1] = zero_one(range_nm2(L->Phi_dot)*L->d[i*3+1]) ;
      temp=min( zero_one(range_ns1(L->Phi)*L->d[i*3+0]), zero_one(range_nm2(L->Phi_dot)*L
>d[i*3+1]));
```

```
    return(temp);
case 11:
    L->d_weights[i*3+0] = zero_one(range_ns1(L->Phi)*L->d[i*3+0]) ;
    L->d_weights[i*3+1] = zero_one(range_nb2(L->Phi_dot)*L->d[i*3+1]) ;
    temp=min( zero_one(range_ns1(L->Phi)*L->d[i*3+0]), zero_one(range_nb2(L->Phi_dot)*L-
>d[i*3+1]));
    return(temp);
case 12:
    L->d_weights[i*3+0] = zero_one(range_zo1(L->Phi)*L->d[i*3+0]) ;
    L->d_weights[i*3+1] = zero_one(range_zo2(L->Phi_dot)*L->d[i*3+1]) ;
    temp=min( zero_one(range_zo1(L->Phi)*L->d[i*3+0]), zero_one(range_zo2(L->Phi_dot)*L-
>d[i*3+1]));
    return(temp);
case 13:
    L->d_weights[i*3+0] = zero_one(range_ps1(L->Phi)*L->d[i*3+0]) ;
    L->d_weights[i*3+1] = zero_one(range_pb2(L->Phi_dot)*L->d[i*3+1]) ;
    temp=min( zero_one(range_ps1(L->Phi)*L->d[i*3+0]), zero_one(range_pb2(L->Phi_dot)*L-
>d[i*3+1]));
    return(temp);
case 14:
    L->d_weights[i*3+0] = zero_one(range_zo1(L->Phi)*L->d[i*3+0]) ;
    L->d_weights[i*3+1] = zero_one(range_zo2(L->Phi_dot)*L->d[i*3+1]) ;
    temp=min( zero_one(range_zo1(L->Phi)*L->d[i*3+0]), zero_one(range_zo2(L->Phi_dot)*L-
>d[i*3+1]));
    return(temp);
case 15:
    L->d_weights[i*3+0] = zero_one(range_ps1(L->Phi)*L->d[i*3+0]) ;
    L->d_weights[i*3+1] = zero_one(range_pb2(L->Phi_dot)*L->d[i*3+1]) ;
    temp=min( zero_one(range_ps1(L->Phi)*L->d[i*3+0]), zero_one(range_pb2(L->Phi_dot)*L-
>d[i*3+1]));
    return(temp);
case 16:
    L->d_weights[i*3+0] = zero_one(range_ps1(L->Phi)*L->d[i*3+0]) ;
    L->d_weights[i*3+1] = zero_one(range_zo2(L->Phi_dot)*L->d[i*3+1]) ;
    temp=min( zero_one(range_ps1(L->Phi)*L->d[i*3+0]), zero_one(range_zo2(L->Phi_dot)*L-
>d[i*3+1]));
    return(temp);
case 17:
    L->d_weights[i*3+0] = zero_one(range_zo1(L->Phi)*L->d[i*3+0]) ;
    L->d_weights[i*3+1] = zero_one(range_pb2(L->Phi_dot)*L->d[i*3+1]) ;
    temp=min( zero_one(range_zo1(L->Phi)*L->d[i*3+0]), zero_one(range_pb2(L->Phi_dot)*L-
>d[i*3+1]));
    return(temp);
case 18:
    L->d_weights[i*3+0] = zero_one(range_zo1(L->Phi)*L->d[i*3+0]) ;
    L->d_weights[i*3+1] = zero_one(range_zo2(L->Phi_dot)*L->d[i*3+1]) ;
    temp=min( zero_one(range_zo1(L->Phi)*L->d[i*3+0]), zero_one(range_zo2(L->Phi_dot)*L-
>d[i*3+1]));
    return(temp);
case 19:
    L->d_weights[i*3+0] = zero_one(range_ps1(L->Phi)*L->d[i*3+0]) ;
    L->d_weights[i*3+1] = zero_one(range_pb2(L->Phi_dot)*L->d[i*3+1]) ;
    temp=min( zero_one(range_ps1(L->Phi)*L->d[i*3+0]), zero_one(range_pb2(L->Phi_dot)*L-
>d[i*3+1]));
    return(temp);
case 20:
    L->d_weights[i*3+0] = zero_one(range_ps1(L->Phi)*L->d[i*3+0]) ;
    L->d_weights[i*3+1] = zero_one(range_zo2(L->Phi_dot)*L->d[i*3+1]) ;
    temp=min( zero_one(range_ps1(L->Phi)*L->d[i*3+0]), zero_one(range_zo2(L->Phi_dot)*L-
>d[i*3+1]));
    return(temp);
case 21:
    L->d_weights[i*3+0] = zero_one(range_zo1(L->Phi)*L->d[i*3+0]) ;
    L->d_weights[i*3+1] = zero_one(range_pb2(L->Phi_dot)*L->d[i*3+1]) ;
    temp=min( zero_one(range_zo1(L->Phi)*L->d[i*3+0]), zero_one(range_pb2(L->Phi_dot)*L-
>d[i*3+1]));
    return(temp);
case 22:
    L->d_weights[i*3+0] = zero_one(range_zo1(L->Phi)*L->d[i*3+0]) ;
    L->d_weights[i*3+1] = zero_one(range_zo2(L->Phi_dot)*L->d[i*3+1]) ;
    temp=min( zero_one(range_zo1(L->Phi)*L->d[i*3+0]), zero_one(range_zo2(L->Phi_dot)*L-
>d[i*3+1]));
    return(temp);
case 23:
    L->d_weights[i*3+0] = zero_one(range_ps1(L->Phi)*L->d[i*3+0]) ;
    L->d_weights[i*3+1] = zero_one(range_pb2(L->Phi_dot)*L->d[i*3+1]) ;
    temp=min( zero_one(range_ps1(L->Phi)*L->d[i*3+0]), zero_one(range_pb2(L->Phi_dot)*L-
>d[i*3+1]));
    return(temp);
case 24:
    L->d_weights[i*3+0] = zero_one(range_ps1(L->Phi)*L->d[i*3+0]) ;
    L->d_weights[i*3+1] = zero_one(range_zo2(L->Phi_dot)*L->d[i*3+1]) ;
    temp=min( zero_one(range_ps1(L->Phi)*L->d[i*3+0]), zero_one(range_zo2(L->Phi_dot)*L-
>d[i*3+1]));
    return(temp);
case 25:
    L->d_weights[i*3+0] = zero_one(range_zo1(L->Phi)*L->d[i*3+0]) ;
    L->d_weights[i*3+1] = zero_one(range_pb2(L->Phi_dot)*L->d[i*3+1]) ;
    temp=min( zero_one(range_zo1(L->Phi)*L->d[i*3+0]), zero_one(range_pb2(L->Phi_dot)*L-
>d[i*3+1]));
    return(temp);
case 26:
    L->d_weights[i*3+0] = zero_one(range_zo1(L->Phi)*L->d[i*3+0]) ;
    L->d_weights[i*3+1] = zero_one(range_zo2(L->Phi_dot)*L->d[i*3+1]) ;
    temp=min( zero_one(range_zo1(L->Phi)*L->d[i*3+0]), zero_one(range_zo2(L->Phi_dot)*L-
>d[i*3+1]));
    return(temp);
}
}
```

```
temp=min( zero_one(range(3)),
          d[i*3+1][j]);
return(temp);
}
```

```
      return(temp);
    case 11:
       L->d_weights[i*3+0] = zero_one(range_ns1(L->Phi)*L->d[i*3+0]) ;
       L->d_weights[i*3+1] = zero_one(range_nb2(L->Phi_dot)*L->d[i*3+1]) ;
       temp=min( zero_one(range_ns1(L->Phi)*L->d[i*3+0]), zero_one(range_nb2(L->Phi_dot)*L-
>d[i*3+1]));
       return(temp);
    case 12:
       L->d_weights[i*3+0] = zero_one(range_zo1(L->Phi)*L->d[i*3+0]) ;
       L->d_weights[i*3+1] = zero_one(range_zo2(L->Phi_dot)*L->d[i*3+1]) ;
       temp=min( zero_one(range_zo1(L->Phi)*L->d[i*3+0]), zero_one(range_zo2(L->Phi_dot)*L-
>d[i*3+1]));
       return(temp);
    case 13:
       L->d_weights[i*3+0] = zero_one(range_ps1(L->Phi)*L->d[i*3+0]) ;
       L->d_weights[i*3+1] = zero_one(range_pb2(L->Phi_dot)*L->d[i*3+1]) ;
       temp=min( zero_one(range_ps1(L->Phi)*L->d[i*3+0]), zero_one(range_pb2(L->Phi_dot)*L-
>d[i*3+1]));
       return(temp);
    case 14:
       L->d_weights[i*3+0] = zero_one(range_ps1(L->Phi)*L->d[i*3+0]) ;
       L->d_weights[i*3+1] = zero_one(range_pm2(L->Phi_dot)*L->d[i*3+1]) ;
       temp=min( zero_one(range_ps1(L->Phi)*L->d[i*3+0]), zero_one(range_pm2(L->Phi_dot)*L-
>d[i*3+1]));
       return(temp);
    case 15:
       L->d_weights[i*3+0] = zero_one(range_ps1(L->Phi)*L->d[i*3+0]) ;
       L->d_weights[i*3+1] = zero_one(range_ps2(L->Phi_dot)*L->d[i*3+1]) ;
       temp=min( zero_one(range_ps1(L->Phi)*L->d[i*3+0]), zero_one(range_ps2(L->Phi_dot)*L-
>d[i*3+1]));
       return(temp);
    case 16:
       L->d_weights[i*3+0] = zero_one(range_ps1(L->Phi)*L->d[i*3+0]) ;
       L->d_weights[i*3+1] = zero_one(range_zo2(L->Phi_dot)*L->d[i*3+1]) ;
       temp=min( zero_one(range_ps1(L->Phi)*L->d[i*3+0]), zero_one(range_zo2(L->Phi_dot)*L-
>d[i*3+1]));
       return(temp);
    case 17:
       L->d_weights[i*3+0] = zero_one(range_pm1(L->Phi)*L->d[i*3+0]) ;
       L->d_weights[i*3+1] = zero_one(range_pb2(L->Phi_dot)*L->d[i*3+1]) ;
       temp=min( zero_one(range_pm1(L->Phi)*L->d[i*3+0]), zero_one(range_pb2(L->Phi_dot)*L-
>d[i*3+1]));
       return(temp);
    case 18:
       L->d_weights[i*3+0] = zero_one(range_pm1(L->Phi)*L->d[i*3+0]) ;
       L->d_weights[i*3+1] = zero_one(range_pm2(L->Phi_dot)*L->d[i*3+1]) ;
       temp=min( zero_one(range_pm1(L->Phi)*L->d[i*3+0]), zero_one(range_pm2(L->Phi_dot)*L-
>d[i*3+1]));
       return(temp);
    case 19:
       L->d_weights[i*3+0] = zero_one(range_pm1(L->Phi)*L->d[i*3+0]) ;
       L->d_weights[i*3+1] = zero_one(range_ps2(L->Phi_dot)*L->d[i*3+1]) ;
       temp=min( zero_one(range_pm1(L->Phi)*L->d[i*3+0]), zero_one(range_ps2(L->Phi_dot)*L-
>d[i*3+1]));
       return(temp);
    case 20:
       L->d_weights[i*3+0] = zero_one(range_pm1(L->Phi)*L->d[i*3+0]) ;
       L->d_weights[i*3+1] = zero_one(range_zo2(L->Phi_dot)*L->d[i*3+1]) ;
       temp=min( zero_one(range_pm1(L->Phi)*L->d[i*3+0]), zero_one(range_zo2(L->Phi_dot)*L-
>d[i*3+1]));
       return(temp);
    case 21:
       L->d_weights[i*3+0] = zero_one(range_pb1(L->Phi)*L->d[i*3+0]) ;
       L->d_weights[i*3+1] = zero_one(range_pb2(L->Phi_dot)*L->d[i*3+1]) ;
```

```
      temp=min( zero_one(range_pb1(L->Phi)*L->d[i*3+0]), zero_one(range_pb2(L->Phi_dot)*L
>d[i*3+1]));
      return(temp);
   case 22:
      L->d_weights[i*3+0] = zero_one(range_pb1(L->Phi)*L->d[i*3+0]) ;
      L->d_weights[i*3+1] = zero_one(range_pm2(L->Phi_dot)*L->d[i*3+1]) ;
      temp=min( zero_one(range_pb1(L->Phi)*L->d[i*3+0]), zero_one(range_pm2(L->Phi_dot)*L-
>d[i*3+1]));
      return(temp);
   case 23:
      L->d_weights[i*3+0] = zero_one(range_pb1(L->Phi)*L->d[i*3+0]) ;
      L->d_weights[i*3+1] = zero_one(range_ps2(L->Phi_dot)*L->d[i*3+1]) ;
      temp=min( zero_one(range_pb1(L->Phi)*L->d[i*3+0]), zero_one(range_ps2(L->Phi_dot)*L
>d[i*3+1]));
      return(temp);
   case 24:
      L->d_weights[i*3+0] = zero_one(range_pb1(L->Phi)*L->d[i*3+0]) ;
      L->d_weights[i*3+1] = zero_one(range_zo2(L->Phi_dot)*L->d[i*3+1]) ;
      temp=min( zero_one(range_pb1(L->Phi)*L->d[i*3+0]), zero_one(range_zo2(L->Phi_dot)*L
>d[i*3+1]));
      return(temp);
  }
}

double range_calculate_z_array(i, L)
int i;
LEARN_CYCLE * L;        /*IN : */
{
  switch (i) {
    case 0:
      return(range_pm3(L->w[0]));
    case 1:
      return(range_pb3(L->w[1]));
    case 2:
      return(range_pb3(L->w[2]));
    case 3:
      return(range_pb3(L->w[3]));
    case 4:
      return(range_ps3(L->w[4]));
    case 5:
      return(range_pm3(L->w[5]));
    case 6:
      return(range_pb3(L->w[6]));
    case 7:
      return(range_pb3(L->w[7]));
    case 8:
      return(range_ps3(L->w[8]));
    case 9:
      return(range_ps3(L->w[9]));
    case 10:
      return(range_pm3(L->w[10]));
    case 11:
      return(range_pb3(L->w[11]));
    case 12:
      return(range_zo3(L->w[12]));
    case 13:
      return(range_nb3(L->w[13]));
    case 14:
      return(range_nm3(L->w[14]));
    case 15:
      return(range_ns3(L->w[15]));
    case 16:
      return(range_ns3(L->w[16]));
    case 17:
```

```
        return(range_nb3(L->w[17]));
      case 18:
        return(range_nb3(L->w[18]));
      case 19:
        return(range_nm3(L->w[19]));
      case 20:
        return(range_ns3(L->w[20]));
      case 21:
        return(range_nb3(L->w[21]));
      case 22:
        return(range_nb3(L->w[22]));
      case 23:
        return(range_nb3(L->w[23]));
      case 24:
        return(range_nm3(L->w[24]));

    }
  }
```

A2. Source code for elevation control

```
#include <stdio.h>
#include <math.h>
#include <sys/types.h>

        /* EXTERNAL DATA STRUCTURE DEFINITION */
#include "../orb_fuzzy/learn_cycle.h"
#define max(x,y)      ( (x >= y) ? x : y )
#define min(x,y)      ( (x < y) ? x : y )
#define Gamma         0.9
#define Beta          0.2
#define Beta_h        0.05
#define Rho           1.0
/* 1 July 92 Change Rho_h from 0.2 to 0.8 */
#define Rho_h         0.8
#define Rho1          2.0
#define Rho_h1        0.4
extern double sgn();
extern double exp();
extern double rnd();
extern double sim_time();

learn_elev(L)
LEARN_CYCLE * L;        /*IN : */
{
  int i,i1, j, k;
  double elev_match(), elev_calculate_z_array();

  double temp ;


        L->x[0] = (L->Phi+40)/80 ;
        L->x[1] = (L->Phi_dot+10)/20 ;


     /* CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC */
     /* 1 July 1992 - Set Bias to 0.0             */
     /* CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC */
     L->x[2] = 0.0 ;

     L->failure = 0;

     /* Set up and evaluate the failure criteria */
        if ( (fabs(L->Phi) > 40) || (fabs(L->Phi_dot) > 10) ) {
          L->failure = -1.;
          if( L->learn_flag ) {
            fprintf(stderr,"learn_elev: Failure at %f.\n",sim_time());
          } /* end if */
        } /* end if */

   /* CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC */
   /* CC 6 November, 1992                              CC */
   /* CC Turn off learning when there is a failure     CC */
   /* CC until there has been no failure for 4 pass    CC */
   /* CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC */

   L->fourth_pass_failure = L->third_pass_failure ;
   L->third_pass_failure = L->second_pass_failure ;
   L->second_pass_failure = L->first_pass_failure ;
   L->first_pass_failure = 0 ;
   if(L->failure == -1) {
     L->first_pass_failure = 1 ;
   } /* end if */

   /* If No Failure in the last 4 passes then turn learning on */
```

```
    if( !L->fourth_pass_failure &&
        !L->third_pass_failure  &&
        !L->second_pass_failure  &&
        !L->first_pass_failure ) {
      L->learn_flag = 1 ;
    } /* end if */

    /* If Failure first pass, but not second pass, let the learn
      flag stay on, so that the failure is processed. On the next
      pass (where there is a failure second pass, but not third
      pass) then turn learning off until there are no failures for
      four passes */
    if( L->second_pass_failure && !(L->third_pass_failure) ) {
      L->learn_flag = 0 ;
    } /* end if */


    /* output: state evaluation */

    for (i = 0; i < 31; i++)
      {
        L->sum = 0.0;
        for(j = 0; j < 3; j++)
          {
            L->sum += L->a[i*3+j] * L->x_old[j];
          }
/*CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CC JUNE 12 , 1992 - Change to "learning speed"     CC
CC L->y[i] = 1.0 / (1.0 + exp(-1.0 * L->sum));     CC
CC JULY  1 , 1992 - Change to "learning speed" 1.0 CC
CC L->y[i] = 1.0 / (1.0 + exp(-0.1 * L->sum/3.0)); CC
CC JULY  1 , 1992 - Normalize For Rules Only       CC
CC L->y[i] = 1.0 / (1.0 + exp(-1.0 * L->sum/3.0)); CC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC */
        L->y[i] = 1.0 / (1.0 + exp(-1.0 * L->sum));
      }
    L->sum = 0.0;
    for(i = 0; i < 31; i++)
      {
        L->sum += L->c[i] * L->y[i];
      }
    L->sum1 = 0.0;
    for ( j = 0;j < 3; j++)
      {
        L->sum1 += L->b[j] * L->x_old[j];
      }
    L->v = L->sum + L->sum1;

    /* output: action */
    for(i = 0; i < 31; i++)
      {
          i1=i;

        L->w[i] = elev_match(i1,L);
        L->z1[i] = elev_calculate_z_array(i1,L);
      }
    L->num1 = 0.0;
    L->denom = 0.0;
    for(i = 0; i < 31; i++)
      {
        L->num1 += L->w[i] * L->z1[i] * L->f[i] ;
        L->denom += L->w[i]*L->f[i] ;

      }
```

```
    /* JUNE 9, 1992 - CORRECTION !!! */
    /* Add test for denom very small compared to sum1 - no rule firing zone */
    /* L->push = (1000.0*fabs(L->denom)<fabs(L->sum1))?0.0:L->sum1/L->denom;*/
    if(fabs(L->num1)<=0.01 || fabs(L->denom)<=0.01) {
      L->push = 0.0 ;
    } else {
      L->push = L->num1/L->denom ;
    } /* end if */

    /* output: action computations completed */
    for(i = 0; i < 31; i++)
      {

        L->sum = 0.0;

        for (j = 0; j < 3; j++)
          L->sum += L->d[i*3+j] * L->x_old[j];

  /*CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
  CC JUNE 12 , 1992 - Change to "learning speed"      CC
  CC L->z[i] = 1.0 / (1.0 + exp(-1.0 * L->sum));      CC
  CC JULY  1 , 1992 - Change to "learning speed"      CC
  CC L->z[i] = 1.0 / (1.0 + exp(-0.1 * L->sum/3.0)); CC
  CC JULY  1 , 1992 - Normalize For Rules Only        CC
  CC L->z[i] = 1.0 / (1.0 + exp(-1.0 * L->sum/3.0)); CC
  CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC*/
          L->z[i] = 1.0 / (1.0 + exp(-1.0 * L->sum));
      }
    L->sum2 = 0.0;
    L->sum3 = 0.0;

    for(i = 0; i < 3; i++)
      L->sum2 += L->e[i] * L->x_old[i];

    for (i=0;i < 31; i++)
      L->sum3 += L->f[i] * L->z[i];

  /*CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
  CC JULY 1 , 1992 - Change Normalize of sum4         CC
  CC L->sum4 = L->sum3 + L->sum2;                     CC
  CC JULY 1 , 1992 - Normalize of sum3 by # rules     CC
  CC L->sum4 = L->sum3 / 31.0 + L->sum2 / 3.0 ;
  CC OCT 2 , 1992 - Do NOT normalize for # rules      CC
  CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC*/

  L->sum4 = L->sum3 + L->sum2 / 3.0 ;

  /*CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
  CC JUNE 12 , 1992 - Change to "learning speed"      CC
  CC L->p = 1.0 / (1.0 + exp(-1.0 * L->sum4));        CC
  CC JULY  1 , 1992 - Change to "learning speed" 1.0 CC
  CC L->p = 1.0 / (1.0 + exp(-0.1 * L->sum4/34.0));  CC
  CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC*/
    L->p = 1.0 / (1.0 + exp(-1.0 * L->sum4));

    /* CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC */
    /* 15 April 1992 - Use temp variable - not push          */
    /* CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC */
    /* L->push = (rnd() <= L->p) ? L->push : -L->push;       */
    /* L->unusualness = (L->push > 0) ? 1.0 - L->p : -L->p; */

    /* CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC */
    /* 28 Sept 1992 - Set stocastic action to zero out cmd   */
```

```
  /* ccccccccccccccccccccccccccccccccccccccccccccccccccccc */
  /* L->push = ( rnd() <= ((L->p+1.0)/2.0) ) ? L->push : -L->push; */

/* ccccccccccccccccccccccccccccccccccccccccccccccccccccc
   28 Sept 1992 - Change definition of unusualness to
   correspond with Hamid's notes.
L->push = ( rnd() <= ((L->p+1.0)/2.0) ) ? L->push : 0.0 ;
L->unusualness = (L->push > 0) ? 1.0 - L->p : -L->p;
*/
  if( rnd() <= ((L->p+1.0)/2.0) ) {
    L->unusualness = 1.0 - L->p ;
  } else {
    L->push = 0.0 ;
    L->unusualness = -L->p ;
  } /* end if */


  /*   using new input values and unmodified weights. */
  /*   Use y_new and v_new so not to destroy y and v. */
  for(i = 0; i < 31; i++)
    {
      L->sum = 0.0;
      for(j = 0; j < 3; j++)
        {
          L->sum += L->a[i*3+j] * L->x[j];
        }
/*ccccccccccccccccccccccccccccccccccccccccccccccccccccc
CC JUNE 12 , 1992 - Change to "learning speed"     CC
CC L->y_new[i] = 1.0 / (1.0 + exp(-1.0 * L->sum)); CC
CC JULY 1  , 1992 - Change to "learning speed"     CC
CC L->y_new[i] = 1.0 / (1.0 + exp(-0.1 * L->sum/3.0));CC
CC JULY 1 , 1992 - Do Not Normalize  sum             CC
CC L->y_new[i] = 1.0 / (1.0 + exp(-1.0 * L->sum/3.0));CC
ccccccccccccccccccccccccccccccccccccccccccccccccccccc*/
      L->y_new[i] = 1.0 / (1.0 + exp(-1.0 * L->sum));
    }
      L->sum = 0.0;
      L->sum1 = 0.0;
      L->sum2 = 0.0;

  for(j = 0; j < 3; j++)
      L->sum1 += L->b[j] * L->x[j];

  for(i = 0; i < 31; i++)
      L->sum2 += L->c[i] * L->y_new[i];

  L->sum = L->sum1 + L->sum2;
  L->v_new = L->sum;


  /*ccccccccccccccccccccccccccccccccccccccccccccccccccccc*/
  /* 17 September, 1992                                  */
  /* This logic depends upon a "crisp" (two valued) failure  */
  /* It has been replaced by "fuzzy" failures            */
  /* action evaluation */
      if (L->failure)
        L->r_hat = L->failure - L->v;
      else
        L->r_hat = L->failure + Gamma * L->v_new - L->v;
  /*ccccccccccccccccccccccccccccccccccccccccccccccccccccc*/
/*  L->r_hat = -1.0 * L->failure +
/*             ( 1.0 - L->failure ) * Gamma * L->v_new - L->v ;

  /* modification and update to parameters */
```

```
      for(i = 0; i < 31; i++)
        {
          L->factor1 = Beta_h * L->r_hat * L->y[i] * (1.0 - L->y[i]) * sgn(L->c[i]);
          L->c[i] += Beta * L->r_hat * L->y[i];

           for(j = 0; j < 3; j++)
                {
                  L->a[i*3+j] += L->factor1 * L->x_old[j];
                }
         }


      for(i = 0; i < 3; i++)
              L->b[i] += Beta * L->r_hat * L->x_old[i];

      for(i = 0; i < 31; i++)
        {
/*CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CC 7 July, 1992 : Weight updates of D's by rule firing strength CC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC*/
          L->factor2 = Rho_h * L->r_hat * L->z[i] * (1.0 - L->z[i]) * sgn(L->f[i])*L->unus
ualness ;

          for(j = 0; j < 3; j++) {
            if( L->learn_flag ) {
              L->d[i*3+j] += L->factor2 * L->x_old[j] * L->d_weights[i*3+j] ;
            } /* end if */
          } /* end for */

        }


      for(i = 0; i < 31; i++)
{

/*CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CC July 2, 1992 : Include Rule Firing Weight in F update          CC
CC L->f[i] += Rho * L->r_hat * L->unusualness * L->z[i];          CC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC*/
        if( L->learn_flag )  {
           L->f[i] += Rho * L->r_hat * L->unusualness * L->z[i] * L->w[i] ;
        } /* end if */
 }
      for(i = 0; i < 3; i++)
{

        L->e[i] += Rho * L->r_hat * L->unusualness * L->x_old[i];
 }
        L->x_old[0] = L->x[0];
        L->x_old[1] = L->x[1];
        L->x_old[2] = L->x[2];
}
/*************************************************************************/

double sgn(x)
   double x;
{
  if (x < 0.0)
    return (-1.0);
  else if (x > 0.0)
    return (1.0);
  else
    return (0.0);
```

```
}

/* zero_one function returns 0 for negative numbers
                            1 for values > 1
                            x for values between 0 and 1 */
double zero_one(x)

  double x;
{
  if (x < 0)  return (0.0);
  else if (x > 1) return (1.0);
  else return (x);
}

/*********** Membership Function for Elev_Angle *******************************/

double elev_nm1(x)
double x;
{
  return (min( max(( -x-1)/7 , 0.0 ), 1.0 ));
  }
double elev_ns1(x)
double x;
{
  if (x <= -1.0)     return (min( max( (x+8.0)/7 , 0.0 ), 1.0 ));
   else  return (min( max( -x/1 , 0.0 ), 1.0 ));
  }
double elev_zo1(x)
double x;
{
  if (x <= 0)     return (min( max( x+1/1 , 0.0 ), 1.0 ));
  else  return (min( max( (-x+16)/16 , 0.0 ), 1.0 ));
  }
double elev_ps1(x)
double x;
{
  if (x <= 16)     return (min( max( x/16 , 0.0 ), 1.0 ));
  else  return (min( max( (-x+32)/16 , 0.0 ), 1.0 ));
  }
double elev_pm1(x)
double x;
{
  return (min( max(( x-16)/16 , 0.0 ), 1.0 ));
  }

/******************** Elev_Rate Membership Functions ***************************/
double elev_nm2(x)
double x;
{
  return (min( max(( -x-.5)/.5 , 0.0 ), 1.0 ));
  }
double elev_ns2(x)
double x;
{
  if (x <= -.5)     return (min( max(( x+1)/.5 , 0.0 ), 1.0 ));
  else  return (min( max(( -x/.5) , 0.0 ), 1.0 ));
  }
double elev_zo2(x)
double x;
{
  if (x <= 0)     return (min( max(( x+.5)/.5 , 0.0 ), 1.0 ));
  else  return (min( max(( -x+.5)/.5 , 0.0 ), 1.0 ));
```

```
  }
double elev_ps2(x)
double x;
{
  if (x <= .5)      return (min( max(x/.5 , 0.0 ), 1.0 ));
  else  return (min( max(( -x+1)/.5 , 0.0 ), 1.0 ));
  }
double elev_pm2(x)
double x;
{
  return (min( max(( x-.5)/1.5 , 0.0 ), 1.0 ));
  }

/***************** Defuzzification Process with Accel Membership Functions ************/

double elev_nm3(x)
double x;
{
  return(-0.1 - 0.1*x);
}

double elev_ns3(x)
double x;
{
  return(-0.1*x);
}

double elev_zo3(x)
double x;
{
  return(0.0);
}

double elev_ps3(x)
double x;
{
  return(0.1*x);
}

double elev_pm3(x)
double x;
{
  return(0.1 + 0.1*x);
}


double elev_match(i,L)
int i;
LEARN_CYCLE *L;         /*IN : */
{
double temp;
  switch (i) {

   case 0:
      L->d_weights[i*3+0] = zero_one(elev_nm1(L->Phi)*L->d[i*3+0]) ;
      L->d_weights[i*3+1] = zero_one(elev_zo2(L->Phi_dot)*L->d[i*3+1]) ;
      temp=min( zero_one(elev_nm1(L->Phi)*L->d[i*3+0]), zero_one(elev_zo2(L->Phi_dot)*L->d
[i*3+1]));
      return(temp);
   case 1:
      L->d_weights[i*3+0] = zero_one(elev_nm1(L->Phi)*L->d[i*3+0]) ;
      L->d_weights[i*3+1] = zero_one(elev_ns2(L->Phi_dot)*L->d[i*3+1]) ;
      temp=min( zero_one(elev_nm1(L->Phi)*L->d[i*3+0]), zero_one(elev_ns2(L->Phi_dot)*L->d
[i*3+1]));
```

```
      return(temp);
   case 2:
      L->d_weights[i*3+0] = zero_one(elev_nm1(L->Phi)*L->d[i*3+0]) ;
      L->d_weights[i*3+1] = zero_one(elev_nm2(L->Phi_dot)*L->d[i*3+1]) ;
      temp=min( zero_one(elev_nm1(L->Phi)*L->d[i*3+0]), zero_one(elev_nm2(L->Phi_dot)*L->
[i*3+1]));
      return(temp);
   case 3:
      L->d_weights[i*3+0] = zero_one(elev_ns1(L->Phi)*L->d[i*3+0]) ;
      L->d_weights[i*3+1] = zero_one(elev_ps2(L->Phi_dot)*L->d[i*3+1]) ;
      temp=min( zero_one(elev_ns1(L->Phi)*L->d[i*3+0]), zero_one(elev_ps2(L->Phi_dot)*L->d
[i*3+1]));
      return(temp);
   case 4:
      L->d_weights[i*3+0] = zero_one(elev_ns1(L->Phi)*L->d[i*3+0]) ;
      L->d_weights[i*3+1] = zero_one(elev_zo2(L->Phi_dot)*L->d[i*3+1]) ;
      temp=min( zero_one(elev_ns1(L->Phi)*L->d[i*3+0]), zero_one(elev_zo2(L->Phi_dot)*L->
[i*3+1]));
      return(temp);
   case 5:
      L->d_weights[i*3+0] = zero_one(elev_ns1(L->Phi)*L->d[i*3+0]) ;
      L->d_weights[i*3+1] = zero_one(elev_ns2(L->Phi_dot)*L->d[i*3+1]) ;
      temp=min( zero_one(elev_ns1(L->Phi)*L->d[i*3+0]), zero_one(elev_ns2(L->Phi_dot)*L->
[i*3+1]));
      return(temp);
   case 6:
      L->d_weights[i*3+0] = zero_one(elev_ns1(L->Phi)*L->d[i*3+0]) ;
      L->d_weights[i*3+1] = zero_one(elev_nm2(L->Phi_dot)*L->d[i*3+1]) ;
      temp=min( zero_one(elev_ns1(L->Phi)*L->d[i*3+0]), zero_one(elev_nm2(L->Phi_dot)*L->d
[i*3+1]));
      return(temp);
   case 7:
      L->d_weights[i*3+0] = zero_one(elev_zo1(L->Phi)*L->d[i*3+0]) ;
      L->d_weights[i*3+1] = zero_one(elev_pm2(L->Phi_dot)*L->d[i*3+1]) ;
      temp=min( zero_one(elev_zo1(L->Phi)*L->d[i*3+0]), zero_one(elev_pm2(L->Phi_dot)*L->
[i*3+1]));
      return(temp);
   case 8:
      L->d_weights[i*3+0] = zero_one(elev_zo1(L->Phi)*L->d[i*3+0]) ;
      L->d_weights[i*3+1] = zero_one(elev_ps2(L->Phi_dot)*L->d[i*3+1]) ;
      temp=min( zero_one(elev_zo1(L->Phi)*L->d[i*3+0]), zero_one(elev_ps2(L->Phi_dot)*L->d
[i*3+1]));
      return(temp);
   case 9:
      L->d_weights[i*3+0] = zero_one(elev_zo1(L->Phi)*L->d[i*3+0]) ;
      L->d_weights[i*3+1] = zero_one(elev_zo2(L->Phi_dot)*L->d[i*3+1]) ;
      temp=min( zero_one(elev_zo1(L->Phi)*L->d[i*3+0]), zero_one(elev_zo2(L->Phi_dot)*L->
[i*3+1]));
      return(temp);
   case 10:
      L->d_weights[i*3+0] = zero_one(elev_zo1(L->Phi)*L->d[i*3+0]) ;
      L->d_weights[i*3+1] = zero_one(elev_ns2(L->Phi_dot)*L->d[i*3+1]) ;
      temp=min( zero_one(elev_zo1(L->Phi)*L->d[i*3+0]), zero_one(elev_ns2(L->Phi_dot)*L->
[i*3+1]));
      return(temp);
   case 11:
      L->d_weights[i*3+0] = zero_one(elev_zo1(L->Phi)*L->d[i*3+0]) ;
      L->d_weights[i*3+1] = zero_one(elev_nm2(L->Phi_dot)*L->d[i*3+1]) ;
      temp=min( zero_one(elev_zo1(L->Phi)*L->d[i*3+0]), zero_one(elev_nm2(L->Phi_dot)*L->d
[i*3+1]));
      return(temp);
   case 12:
      L->d_weights[i*3+0] = zero_one(elev_ps1(L->Phi)*L->d[i*3+0]) ;
      L->d_weights[i*3+1] = zero_one(elev_pm2(L->Phi_dot)*L->d[i*3+1]) ;
```

```
      temp=min( zero_one(elev_ps1(L->Phi)*L->d[i*3+0]), zero_one(elev_pm2(L->Phi_dot)*L->d
[i*3+1]));
      return(temp);
   case 13:
      L->d_weights[i*3+0] = zero_one(elev_ps1(L->Phi)*L->d[i*3+0]) ;
      L->d_weights[i*3+1] = zero_one(elev_ps2(L->Phi_dot)*L->d[i*3+1]) ;
      temp=min( zero_one(elev_ps1(L->Phi)*L->d[i*3+0]), zero_one(elev_ps2(L->Phi_dot)*L->d
[i*3+1]));
      return(temp);
   case 14:
      L->d_weights[i*3+0] = zero_one(elev_ps1(L->Phi)*L->d[i*3+0]) ;
      L->d_weights[i*3+1] = zero_one(elev_zo2(L->Phi_dot)*L->d[i*3+1]) ;
      temp=min( zero_one(elev_ps1(L->Phi)*L->d[i*3+0]), zero_one(elev_zo2(L->Phi_dot)*L->d
[i*3+1]));
      return(temp);
   case 15:
      L->d_weights[i*3+0] = zero_one(elev_ps1(L->Phi)*L->d[i*3+0]) ;
      L->d_weights[i*3+1] = zero_one(elev_ns2(L->Phi_dot)*L->d[i*3+1]) ;
      temp=min( zero_one(elev_ps1(L->Phi)*L->d[i*3+0]), zero_one(elev_ns2(L->Phi_dot)*L->d
[i*3+1]));
      return(temp);
   case 16:
      L->d_weights[i*3+0] = zero_one(elev_pm1(L->Phi)*L->d[i*3+0]) ;
      L->d_weights[i*3+1] = zero_one(elev_pm2(L->Phi_dot)*L->d[i*3+1]) ;
      temp=min( zero_one(elev_pm1(L->Phi)*L->d[i*3+0]), zero_one(elev_pm2(L->Phi_dot)*L->d
[i*3+1]));
      return(temp);
   case 17:
      L->d_weights[i*3+0] = zero_one(elev_pm1(L->Phi)*L->d[i*3+0]) ;
      L->d_weights[i*3+1] = zero_one(elev_ps2(L->Phi_dot)*L->d[i*3+1]) ;
      temp=min( zero_one(elev_pm1(L->Phi)*L->d[i*3+0]), zero_one(elev_ps2(L->Phi_dot)*L->d
[i*3+1]));
      return(temp);
   case 18:
      L->d_weights[i*3+0] = zero_one(elev_pm1(L->Phi)*L->d[i*3+0]) ;
      L->d_weights[i*3+1] = zero_one(elev_zo2(L->Phi_dot)*L->d[i*3+1]) ;
      temp=min( zero_one(elev_pm1(L->Phi)*L->d[i*3+0]), zero_one(elev_zo2(L->Phi_dot)*L->d
[i*3+1]));
      return(temp);
  }
}

double elev_calculate_z_array(i, L)
int i;
LEARN_CYCLE * L;        /*IN : */
{
  switch (i) {
    case 0:
      return(elev_pm3(L->w[0]));
    case 1:
      return(elev_pm3(L->w[1]));
    case 2:
      return(elev_pm3(L->w[2]));
    case 3:
      return(elev_zo3(L->w[3]));
    case 4:
      return(elev_ps3(L->w[4]));
    case 5:
      return(elev_ps3(L->w[5]));
    case 6:
      return(elev_pm3(L->w[6]));
    case 7:
      return(elev_nm3(L->w[7]));
    case 8:
```

```
        return(elev_ns3(L->w[8]));
      case 9:
        return(elev_zo3(L->w[9]));
      case 10:
        return(elev_ps3(L->w[10]));
      case 11:
        return(elev_pm3(L->w[11]));
      case 12:
        return(elev_nm3(L->w[12]));
      case 13:
        return(elev_ns3(L->w[13]));
      case 14:
        return(elev_ns3(L->w[14]));
      case 15:
        return(elev_zo3(L->w[15]));
      case 16:
        return(elev_nm3(L->w[16]));
      case 17:
        return(elev_nm3(L->w[17]));
      case 18:
        return(elev_ns3(L->w[18]));

    }
}
```

A3. Source code for azimuth control

```
#include <stdio.h>
#include <math.h>
#include <sys/types.h>

        /* EXTERNAL DATA STRUCTURE DEFINITION */
#include "../orb_fuzzy/learn_cycle.h"
#define max(x,y)      ( (x >= y) ? x : y )
#define min(x,y)      ( (x < y) ? x : y )
#define Gamma         0.9
#define Beta          0.2
#define Beta_h        0.05
#define Rho           1.0
/* 1 July 92 Change Rho_h from 0.2 to 0.8 */
#define Rho_h         0.8
#define Rho1          2.0
#define Rho_h1        0.4
extern double sgn();
extern double exp();
extern double rnd();
extern double sim_time();

learn_azim(L)
LEARN_CYCLE * L;        /*IN : */
{
  int i,i1, j, k;
  double azim_match(), azim_calculate_z_array();

  double temp ;


        /* CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC */
        /* 11 March 1992 - Alter scaling             */
        /* CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC */
        /* L->x[0] = L->Phi/20.0;                    */
        /* L->x[1] = L->Phi_dot/4.0;                 */
        /* CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC */
        /* 08 April 1992 - Alter scaling             */
        /* CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC */
        /* L->x[0] = L->Phi/10.0;                    */
        /* L->x[1] = L->Phi_dot/2.0;                 */
        /* CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC */
        L->x[0] = (L->Phi + 40)/80.0;
        L->x[1] = (L->Phi_dot + 10)/20.0;


      /* CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC */
      /* 08 April 1992 - Alter Bias                */
      /* CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC */
      /* L->x[2] = 1.00 ;                          */
      /* CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC */
      /* 15 April 1992 - Alter Bias  to 0.5        */
      /* CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC */
      /* 1 July 1992 - Alter Bias to 0.0           */
      /* CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC */
      L->x[2] = 0.0 ;

      L->failure = 0;

      /* CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC */
      /* CC 12 August, 1992                       CC */
      /* CC Change failure criteria from 0.7 & 0.07 CC */
      /* CC to be 0.5 and 0.05                    CC */
      /* CC 31 August, 1992                       CC */
      /* CC Change back to 0.7 & 0.07             CC */
```

```
    /* CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC */
    /* Set up and evaluate the failure criteria */
       if ( (fabs(L->Phi) > 2.0) || (fabs(L->Phi_dot) > 0.5) ) {
         L->failure = -1.;
         if( L->learn_flag ) {
           fprintf(stderr,"learn_azim: Failure at %f.\n",sim_time());
         } /* end if */
       } /* end if */

/* CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC */
/* CC 6 November, 1992                              CC */
/* CC Turn off learning when there is a failure     CC */
/* CC until there has been no failure for 4 pass    CC */
/* CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC */

L->fourth_pass_failure = L->third_pass_failure ;
L->third_pass_failure = L->second_pass_failure ;
L->second_pass_failure = L->first_pass_failure ;
L->first_pass_failure = 0 ;
if(L->failure == -1) {
  L->first_pass_failure = 1 ;
} /* end if */

/* If No Failure in the last 4 passes then turn learning on */
if( !L->fourth_pass_failure &&
    !L->third_pass_failure  &&
    !L->second_pass_failure  &&
    !L->first_pass_failure ) {
  L->learn_flag = 1 ;
} /* end if */

/* If Failure first pass, but not second pass, let the learn
  flag stay on, so that the failure is processed. On the next
  pass (where there is a failure second pass, but not third
  pass) then turn learning off until there are no failures for
  four passes */
if( L->second_pass_failure && !(L->third_pass_failure) ) {
  L->learn_flag = 0 ;
} /* end if */


/* output: state evaluation */

for (i = 0; i < 16; i++)
  {
    L->sum = 0.0;
    for(j = 0; j < 3; j++)
      {
        L->sum += L->a[i*3+j] * L->x_old[j];
      }
    L->y[i] = 1.0 / (1.0 + exp(-1.0 * L->sum));
  }
L->sum = 0.0;
for(i = 0; i < 16; i++)
  {
    L->sum += L->c[i] * L->y[i];
  }
L->sum1 = 0.0;
for ( j = 0;j < 3; j++)
  {
    L->sum1 += L->b[j] * L->x_old[j];
  }
L->v = L->sum + L->sum1;
```

```
    /* output: action */
    for(i = 0; i < 16; i++)
      {
          i1=i;

        L->w[i] = azim_match(i1,L);
        L->z1[i] = azim_calculate_z_array(i1,L);
      }
    L->num1 = 0.0;
    L->denom = 0.0;
    for(i = 0; i < 16; i++)
      {
        L->num1 += L->w[i] * L->z1[i] * L->f[i] ;
        L->denom += L->w[i]*L->f[i] ;

      }

    /* JUNE 9, 1992 - CORRECTION !!! */
    /* Add test for denom very small compared to sum1 - no rule firing zone */
    /* L->push = (1000.0*fabs(L->denom)<fabs(L->sum1))?0.0:L->sum1/L->denom;*/
    if(fabs(L->num1)<=0.01 || fabs(L->denom)<=0.01) {
      L->push = 0.0 ;
    } else {
      L->push = L->num1/L->denom ;
    } /* end if */

    /* output: action computations completed */
    for(i = 0; i < 16; i++)
      {

        L->sum = 0.0;

        for (j = 0; j < 3; j++)
          L->sum += L->d[i*3+j] * L->x_old[j];

          L->z[i] = 1.0 / (1.0 + exp(-1.0 * L->sum));
      }
    L->sum2 = 0.0;
    L->sum3 = 0.0;

    for(i = 0; i < 3; i++)
      L->sum2 += L->e[i] * L->x_old[i];

    for (i=0;i < 16; i++)
      L->sum3 += L->f[i] * L->z[i];

  /*CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
  CC JULY 1 , 1992 - Change Normalize of sum4          CC
  CC L->sum4 = L->sum3 + L->sum2;                      CC
  CC JULY 1 , 1992 - Normalize of sum3 by # rules      CC
  CC L->sum4 = L->sum3 / 31.0 + L->sum2 / 3.0 ;
  CC OCT 2 , 1992 - Do NOT normalize for # rules       CC
  CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC*/

  L->sum4 = L->sum3 + L->sum2 / 3.0 ;

  /*CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
  CC JUNE 12 , 1992 - Change to "learning speed"       CC
  CC L->p = 1.0 / (1.0 + exp(-1.0 * L->sum4));         CC
  CC JULY  1 , 1992 - Change to "learning speed" 1.0 CC
  CC L->p = 1.0 / (1.0 + exp(-0.1 * L->sum4/34.0));  CC
  CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC*/
    L->p = 1.0 / (1.0 + exp(-1.0 * L->sum4));
```

```
  /* ccccccccccccccccccccccccccccccccccccccccccccccccccccc */
  /* 15 April 1992 - Use temp variable - not push          */
  /* ccccccccccccccccccccccccccccccccccccccccccccccccccccc */
  /* L->push = (rnd() <= L->p) ? L->push : -L->push;       */
  /* L->unusualness = (L->push > 0) ? 1.0 - L->p : -L->p; */

  /* ccccccccccccccccccccccccccccccccccccccccccccccccccccc */
  /* 28 Sept 1992 - Set stocastic action to zero out cmd   */
  /* ccccccccccccccccccccccccccccccccccccccccccccccccccccc */
  /* L->push = ( rnd() <= ((L->p+1.0)/2.0) ) ? L->push : -L->push; */

/* ccccccccccccccccccccccccccccccccccccccccccccccccccccc
   28 Sept 1992 - Change definition of unusualness to
   correspond with Hamid's notes.
L->push = ( rnd() <= ((L->p+1.0)/2.0) ) ? L->push : 0.0 ;
L->unusualness = (L->push > 0) ? 1.0 - L->p : -L->p;
*/
  if( rnd() <= ((L->p+1.0)/2.0) ) {
    L->unusualness = 1.0 - L->p ;
  } else {
    L->push = 0.0 ;
    L->unusualness = -L->p ;
  } /* end if */


  /*   using new input values and unmodified weights. */
  /*   Use y_new and v_new so not to destroy y and v. */
  for(i = 0; i < 16; i++)
    {
      L->sum = 0.0;
      for(j = 0; j < 3; j++)
        {
          L->sum += L->a[i*3+j] * L->x[j];
        }
      L->y_new[i] = 1.0 / (1.0 + exp(-1.0 * L->sum));
    }
      L->sum = 0.0;
      L->sum1 = 0.0;
      L->sum2 = 0.0;

  for(j = 0; j < 3; j++)
      L->sum1 += L->b[j] * L->x[j];

  for(i = 0; i < 16; i++)
      L->sum2 += L->c[i] * L->y_new[i];

  L->sum = L->sum1 + L->sum2;
  L->v_new = L->sum;


  /*ccccccccccccccccccccccccccccccccccccccccccccccccccccc*/
  /* 17 September, 1992                                   */
  /* This logic depends upon a "crisp" (two valued) failure  */
  /* It has been replaced by "fuzzy" failures             */
  /* action evaluation */
      if (L->failure)
        L->r_hat = L->failure - L->v;
      else
        L->r_hat = L->failure + Gamma * L->v_new - L->v;
  /*ccccccccccccccccccccccccccccccccccccccccccccccccccccc*/
/*  L->r_hat = -1.0 * L->failure +
/*             ( 1.0 - L->failure ) * Gamma * L->v_new - L->v ;

  /* modification and update to parameters */
```

```
      for(i = 0; i < 16; i++)
        {
          L->factor1 = Beta_h * L->r_hat * L->y[i] * (1.0 - L->y[i]) * sgn(L->c[i]);
          L->c[i] += Beta * L->r_hat * L->y[i];

           for(j = 0; j < 3; j++)
                {
                  L->a[i*3+j] += L->factor1 * L->x_old[j];
                }
         }


      for(i = 0; i < 3; i++)
              L->b[i] += Beta * L->r_hat * L->x_old[i];

      for(i = 0; i < 16; i++)
        {
/*CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CC 7 July, 1992 : Weight updates of D's by rule firing strength CC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC*/
          L->factor2 = Rho_h * L->r_hat * L->z[i] * (1.0 - L->z[i]) * sgn(L->f[i])*L->unus
ualness ;

          for(j = 0; j < 3; j++) {
            if( L->learn_flag ) {
              L->d[i*3+j] += L->factor2 * L->x_old[j] * L->d_weights[i*3+j] ;
            } /* end if */
          } /* end for */

        }


      for(i = 0; i < 16; i++)
{

/*CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CC July 2, 1992 : Include Rule Firing Weight in F update          CC
CC L->f[i] += Rho * L->r_hat * L->unusualness * L->z[i];          CC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC*/
        if( L->learn_flag )  {
           L->f[i] += Rho * L->r_hat * L->unusualness * L->z[i] * L->w[i] ;
        } /* end if */
 }
       for(i = 0; i < 3; i++)
{

        L->e[i] += Rho * L->r_hat * L->unusualness * L->x_old[i];
 }
        L->x_old[0] = L->x[0];
        L->x_old[1] = L->x[1];
        L->x_old[2] = L->x[2];
}
/*************************************************************************/

double sgn(x)
   double x;
{
  if (x < 0.0)
    return (-1.0);
  else if (x > 0.0)
    return (1.0);
  else
    return (0.0);
```

```
}

/* zero_one function returns 0 for negative numbers
                            1 for values > 1
                            x for values between 0 and 1 */
double zero_one(x)

  double x;
{
  if (x < 0)  return (0.0);
  else if (x > 1) return (1.0);
  else return (x);
}


/*********** Membership Function for Azim *********************************/

double azim_nm1(x)
double x;
{
  return(min (max( (-x-2)/2, 0.0), 1.0));
}
double azim_ns1(x)
double x;
{
  if (x <= -2.0)      return (min( max( (x+4.0)/2.0 , 0.0 ), 1.0 ));
   else  return (min( max( -x/2 , 0.0 ), 1.0 ));
  }
double azim_zo1(x)
double x;
{
  if (x <= 0)      return (min( max( (x+2)/2 , 0.0 ), 1.0 ));
  else  return (min( max( (-x+2)/2 , 0.0 ), 1.0 ));
  }
double azim_ps1(x)
double x;
{
  if (x <= 2)      return (min( max( x/2 , 0.0 ), 1.0 ));
  else  return (min( max( (-x+4)/2 , 0.0 ), 1.0 ));
  }
double azim_pm1(x)
double x;
{
  return (min( max(( x-2)/2 , 0.0 ), 1.0 ));
  }

/******************* Azim_Rate Membership Functions ***************************/

double azim_nm2(x)
double x;
{
  return (min( max(( -x-0.5)/0.5 , 0.0 ), 1.0 ));
  }
double azim_ns2(x)
double x;
{
  if (x <= -.5)      return (min( max(( x+1)/.5 , 0.0 ), 1.0 ));
  else  return (min( max( -x/.5 , 0.0 ), 1.0 ));
  }
double azim_zo2(x)
double x;
{
  if (x <= 0)      return (min( max(( x+.5)/.5 , 0.0 ), 1.0 ));
```

```
  else  return (min( max(( -x+.5)/.5 , 0.0 ), 1.0 ));
  }
double azim_ps2(x)
double x;
{
  if (x <= .5)      return (min( max(x/.5 , 0.0 ), 1.0 ));
  else  return (min( max(( -x+1)/.5 , 0.0 ), 1.0 ));
  }
double azim_pm2(x)
double x;
{
  return (min( max(( x-.5)/1.5 , 0.0 ), 1.0 ));
  }

/***************** Defuzzification Process with Accel Membership Functions ************/

double azim_nm3(x)
double x;
{
  return(-0.1 - 0.1 * x);
}

double azim_ns3(x)
double x;
{
  return(-0.1*x);
}

double azim_zo3(x)
double x;
{
  return(0.0);
}

double azim_ps3(x)
double x;
{
  return(0.1*x);
}

double azim_pm3(x)
double x;
{
  return(0.1 + 0.1*x);
}


double azim_match(i,L)
int i;
LEARN_CYCLE *L;         /*IN : */
{
double temp;
  switch (i) {

   case 0:
      L->d_weights[i*3+0] = zero_one(azim_nm1(L->Phi)*L->d[i*3+0]) ;
      L->d_weights[i*3+1] = zero_one(azim_zo2(L->Phi_dot)*L->d[i*3+1]) ;
      temp=min( zero_one(azim_nm1(L->Phi)*L->d[i*3+0]), zero_one(azim_zo2(L->Phi_dot)*L->d
[i*3+1]));
      return(temp);
   case 1:
      L->d_weights[i*3+0] = zero_one(azim_nm1(L->Phi)*L->d[i*3+0]) ;
      L->d_weights[i*3+1] = zero_one(azim_ns2(L->Phi_dot)*L->d[i*3+1]) ;
      temp=min( zero_one(azim_nm1(L->Phi)*L->d[i*3+0]), zero_one(azim_ns2(L->Phi_dot)*L->d
```

```
[i*3+1]));
      return(temp);
   case 2:
      L->d_weights[i*3+0] = zero_one(azim_nm1(L->Phi)*L->d[i*3+0]) ;
      L->d_weights[i*3+1] = zero_one(azim_nm2(L->Phi_dot)*L->d[i*3+1]) ;
      temp=min( zero_one(azim_nm1(L->Phi)*L->d[i*3+0]), zero_one(azim_nm2(L->Phi_dot)*L->
[i*3+1]));
      return(temp);
   case 3:
      L->d_weights[i*3+0] = zero_one(azim_ns1(L->Phi)*L->d[i*3+0]) ;
      L->d_weights[i*3+1] = zero_one(azim_zo2(L->Phi_dot)*L->d[i*3+1]) ;
      temp=min( zero_one(azim_ns1(L->Phi)*L->d[i*3+0]), zero_one(azim_zo2(L->Phi_dot)*L->
[i*3+1]));
      return(temp);
   case 4:
      L->d_weights[i*3+0] = zero_one(azim_ns1(L->Phi)*L->d[i*3+0]) ;
      L->d_weights[i*3+1] = zero_one(azim_ns2(L->Phi_dot)*L->d[i*3+1]) ;
      temp=min( zero_one(azim_ns1(L->Phi)*L->d[i*3+0]), zero_one(azim_ns2(L->Phi_dot)*L->d
[i*3+1]));
      return(temp);
   case 5:
      L->d_weights[i*3+0] = zero_one(azim_ns1(L->Phi)*L->d[i*3+0]) ;
      L->d_weights[i*3+1] = zero_one(azim_nm2(L->Phi_dot)*L->d[i*3+1]) ;
      temp=min( zero_one(azim_ns1(L->Phi)*L->d[i*3+0]), zero_one(azim_nm2(L->Phi_dot)*L->
[i*3+1]));
      return(temp);
   case 6:
      L->d_weights[i*3+0] = zero_one(azim_zo1(L->Phi)*L->d[i*3+0]) ;
      L->d_weights[i*3+1] = zero_one(azim_pm2(L->Phi_dot)*L->d[i*3+1]) ;
      temp=min( zero_one(azim_zo1(L->Phi)*L->d[i*3+0]), zero_one(azim_pm2(L->Phi_dot)*L->d
[i*3+1]));
      return(temp);
   case 7:
      L->d_weights[i*3+0] = zero_one(azim_zo1(L->Phi)*L->d[i*3+0]) ;
      L->d_weights[i*3+1] = zero_one(azim_ps2(L->Phi_dot)*L->d[i*3+1]) ;
      temp=min( zero_one(azim_zo1(L->Phi)*L->d[i*3+0]), zero_one(azim_ps2(L->Phi_dot)*L->
[i*3+1]));
      return(temp);
   case 8:
      L->d_weights[i*3+0] = zero_one(azim_zo1(L->Phi)*L->d[i*3+0]) ;
      L->d_weights[i*3+1] = zero_one(azim_zo2(L->Phi_dot)*L->d[i*3+1]) ;
      temp=min( zero_one(azim_zo1(L->Phi)*L->d[i*3+0]), zero_one(azim_zo2(L->Phi_dot)*L->
[i*3+1]));
      return(temp);
   case 9:
      L->d_weights[i*3+0] = zero_one(azim_zo1(L->Phi)*L->d[i*3+0]) ;
      L->d_weights[i*3+1] = zero_one(azim_ns2(L->Phi_dot)*L->d[i*3+1]) ;
      temp=min( zero_one(azim_zo1(L->Phi)*L->d[i*3+0]), zero_one(azim_ns2(L->Phi_dot)*L->d
[i*3+1]));
      return(temp);
   case 10:
      L->d_weights[i*3+0] = zero_one(azim_zo1(L->Phi)*L->d[i*3+0]) ;
      L->d_weights[i*3+1] = zero_one(azim_nm2(L->Phi_dot)*L->d[i*3+1]) ;
      temp=min( zero_one(azim_zo1(L->Phi)*L->d[i*3+0]), zero_one(azim_nm2(L->Phi_dot)*L->
[i*3+1]));
      return(temp);
   case 11:
      L->d_weights[i*3+0] = zero_one(azim_ps1(L->Phi)*L->d[i*3+0]) ;
      L->d_weights[i*3+1] = zero_one(azim_pm2(L->Phi_dot)*L->d[i*3+1]) ;
      temp=min( zero_one(azim_ps1(L->Phi)*L->d[i*3+0]), zero_one(azim_pm2(L->Phi_dot)*L->d
[i*3+1]));
      return(temp);
   case 12:
      L->d_weights[i*3+0] = zero_one(azim_ps1(L->Phi)*L->d[i*3+0]) ;
```

```
        L->d_weights[i*3+1] = zero_one(azim_ps2(L->Phi_dot)*L->d[i*3+1]) ;
        temp=min( zero_one(azim_ps1(L->Phi)*L->d[i*3+0]), zero_one(azim_ps2(L->Phi_dot)*L->d
[i*3+1]));
        return(temp);
      case 13:
        L->d_weights[i*3+0] = zero_one(azim_ps1(L->Phi)*L->d[i*3+0]) ;
        L->d_weights[i*3+1] = zero_one(azim_zo2(L->Phi_dot)*L->d[i*3+1]) ;
        temp=min( zero_one(azim_ps1(L->Phi)*L->d[i*3+0]), zero_one(azim_zo2(L->Phi_dot)*L->d
[i*3+1]));
        return(temp);
      case 14:
        L->d_weights[i*3+0] = zero_one(azim_pm1(L->Phi)*L->d[i*3+0]) ;
        L->d_weights[i*3+1] = zero_one(azim_pm2(L->Phi_dot)*L->d[i*3+1]) ;
        temp=min( zero_one(azim_pm1(L->Phi)*L->d[i*3+0]), zero_one(azim_pm2(L->Phi_dot)*L->d
[i*3+1]));
        return(temp);
      case 15:
        L->d_weights[i*3+0] = zero_one(azim_pm1(L->Phi)*L->d[i*3+0]) ;
        L->d_weights[i*3+1] = zero_one(azim_ps2(L->Phi_dot)*L->d[i*3+1]) ;
        temp=min( zero_one(azim_pm1(L->Phi)*L->d[i*3+0]), zero_one(azim_ps2(L->Phi_dot)*L->d
[i*3+1]));
        return(temp);
      case 16:
        L->d_weights[i*3+0] = zero_one(azim_pm1(L->Phi)*L->d[i*3+0]) ;
        L->d_weights[i*3+1] = zero_one(azim_zo2(L->Phi_dot)*L->d[i*3+1]) ;
        temp=min( zero_one(azim_pm1(L->Phi)*L->d[i*3+0]), zero_one(azim_zo2(L->Phi_dot)*L->d
[i*3+1]));
        return(temp);
    }
}

double azim_calculate_z_array(i, L)
int i;
LEARN_CYCLE * L;        /*IN : */
{
  switch (i) {
    case 0:
      return(azim_pm3(L->w[0]));
    case 1:
      return(azim_pm3(L->w[1]));
    case 2:
      return(azim_pm3(L->w[2]));
    case 3:
      return(azim_ps3(L->w[3]));
    case 4:
      return(azim_ps3(L->w[4]));
    case 5:
      return(azim_pm3(L->w[5]));
    case 6:
      return(azim_nm3(L->w[6]));
    case 7:
      return(azim_ns3(L->w[7]));
    case 8:
      return(azim_zo3(L->w[8]));
    case 9:
      return(azim_ps3(L->w[9]));
    case 10:
      return(azim_pm3(L->w[10]));
    case 11:
      return(azim_nm3(L->w[11]));
    case 12:
      return(azim_ns3(L->w[12]));
    case 13:
      return(azim_ns3(L->w[13]));
```

```
      case 14:
        return(azim_nm3(L->w[14]));
      case 15:
        return(azim_nm3(L->w[15]));
      case 16:
        return(azim_nm3(L->w[16]));

    }
  }
```

Appendix B.

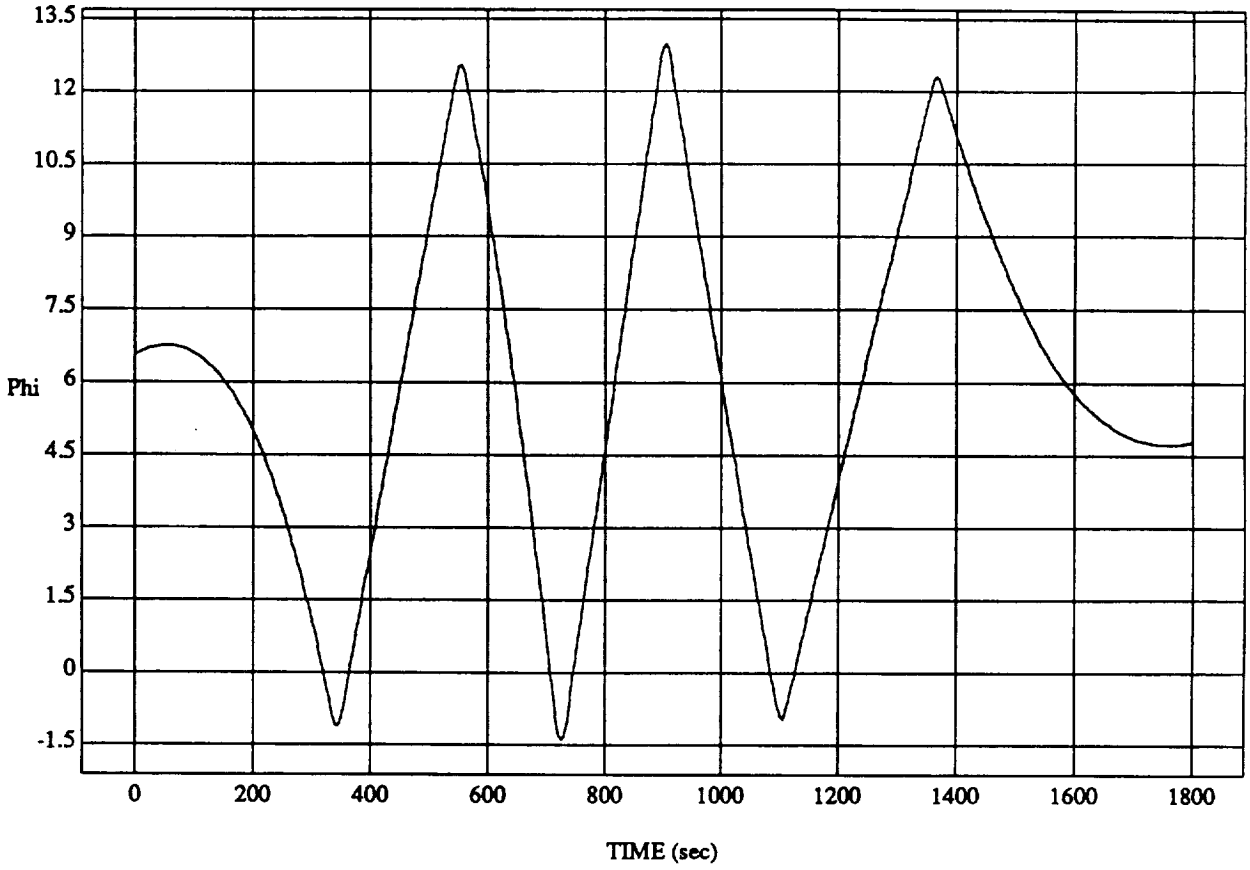
Plots of Selected Parameters
for
Shuttle Translational Control

B1. Shuttle V-bar Approach from 400 feet to 50 feet

SIMULATION APPLICATION: ARIC Translational Controller Simulation

Phi vs TIME

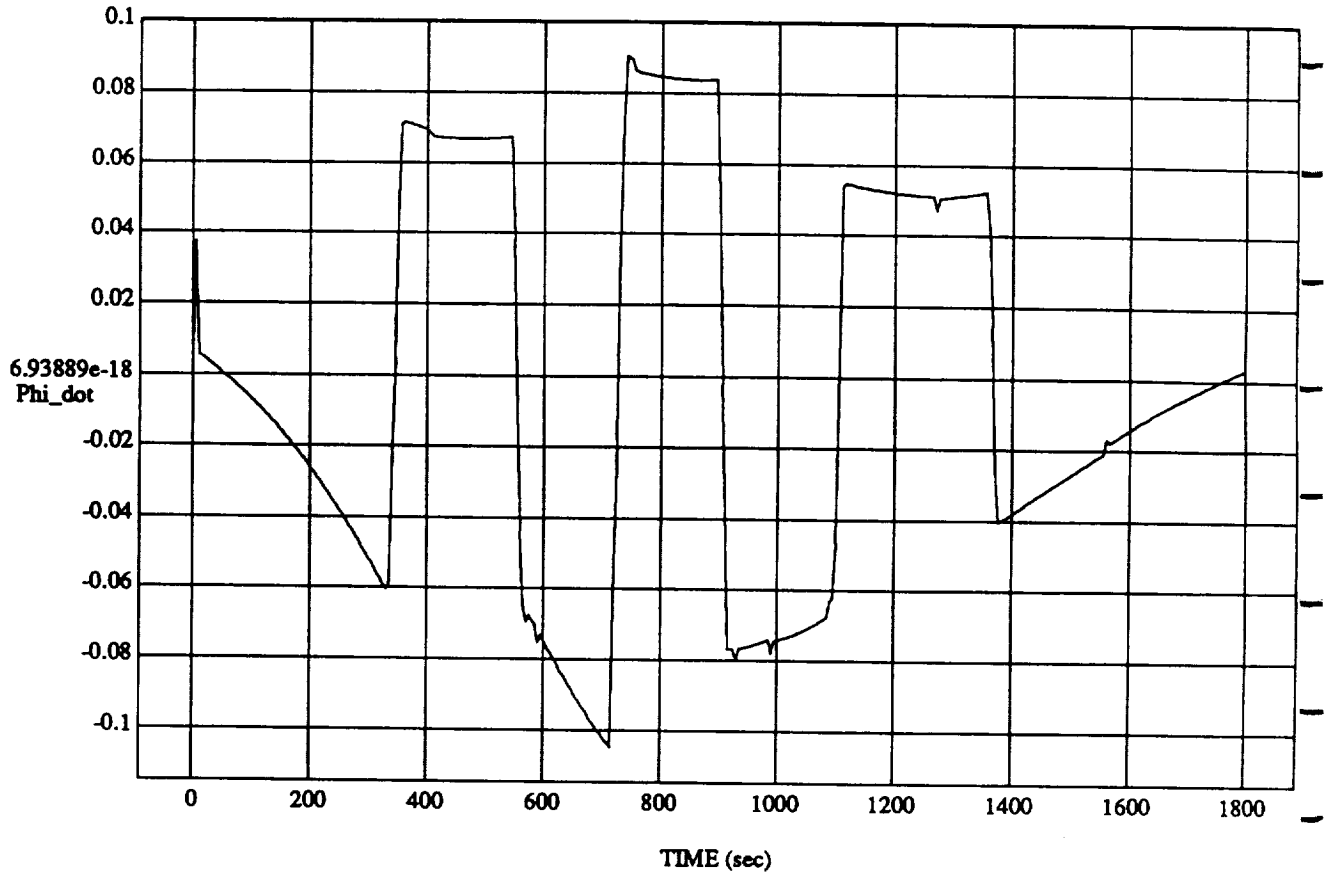
RUN: V Bar Approach



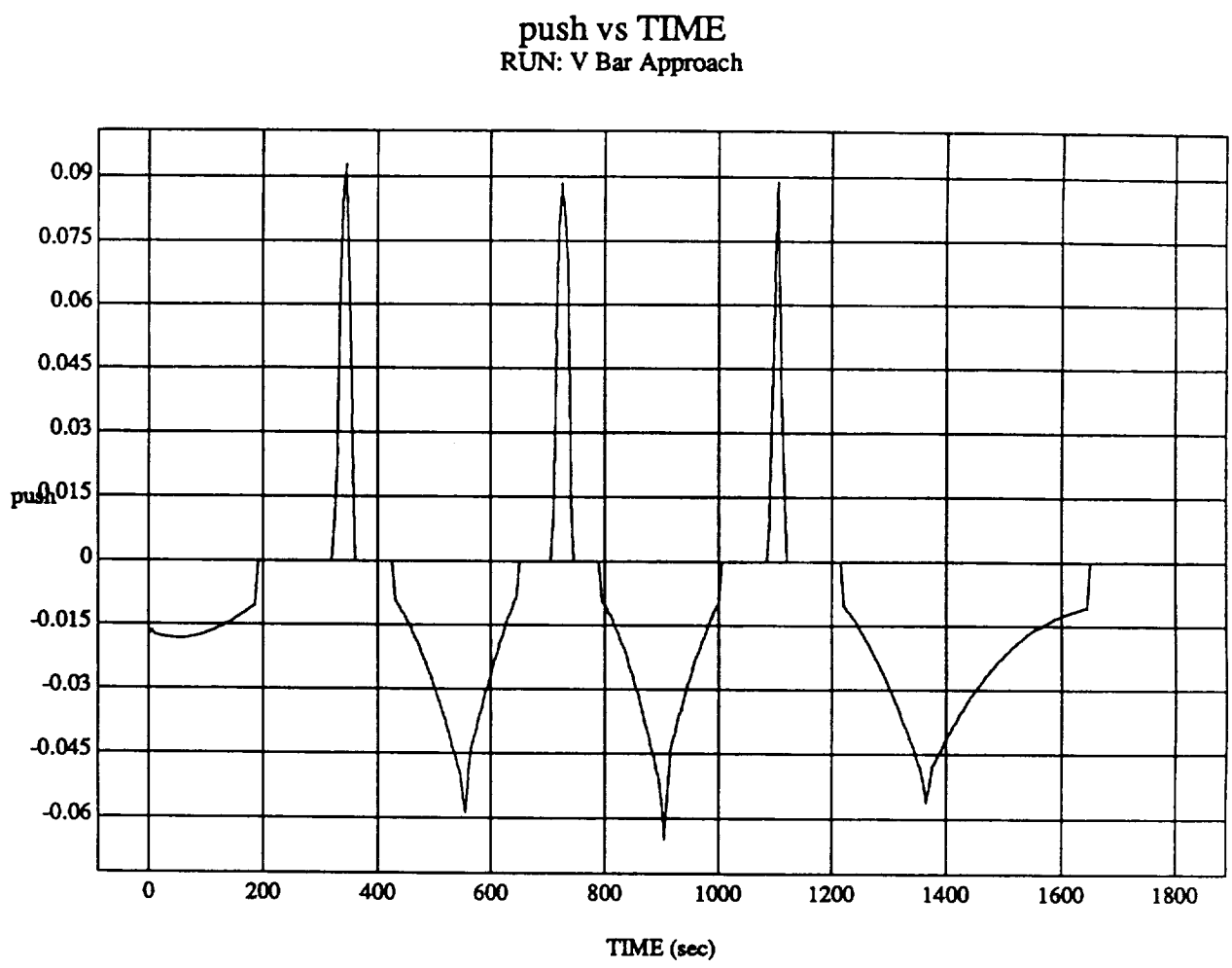
MODULE: ORBITER.lm_elev

DATA SAMPLING FREQUENCY: 0.200 Hz

Phi_dot vs TIME
RUN: V Bar Approach



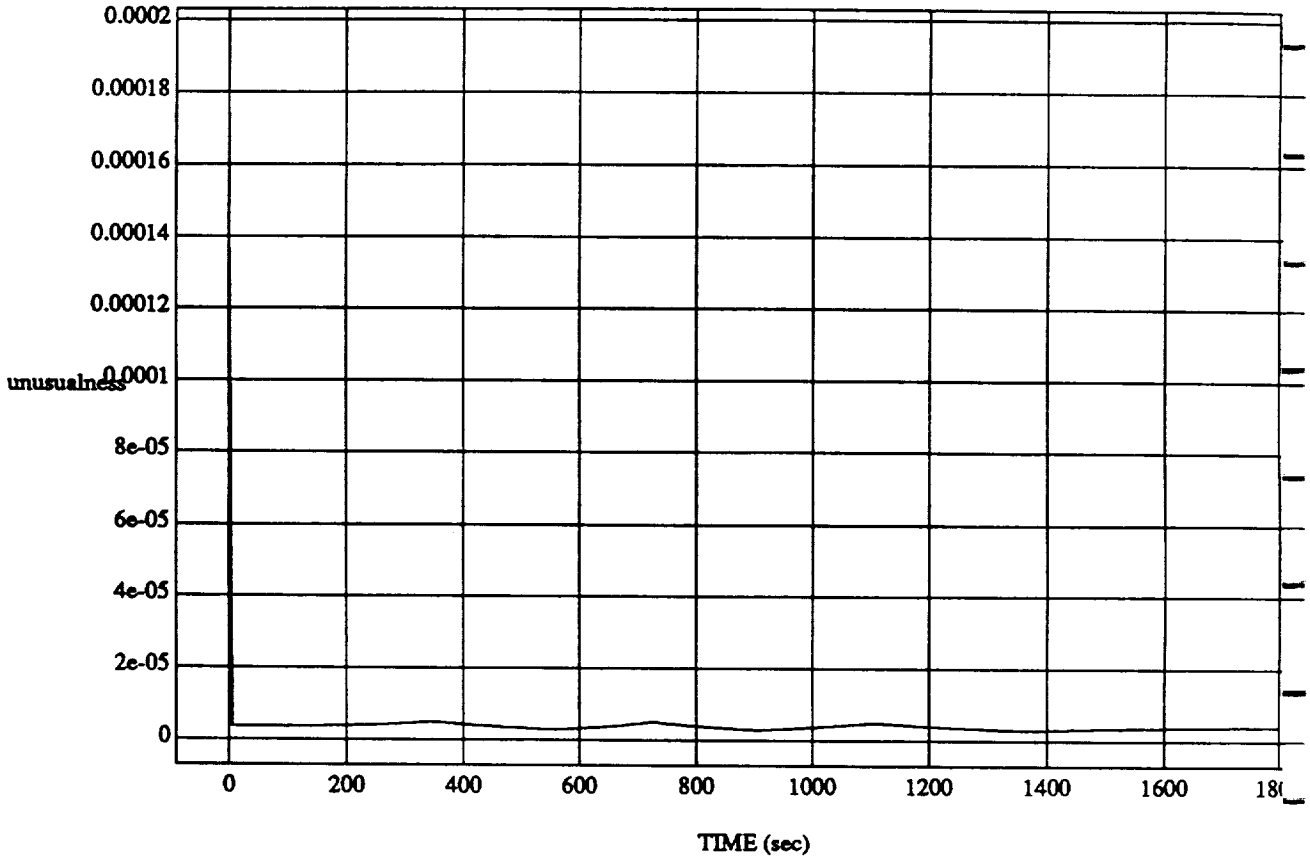
MODULE: ORBITER.lm_elev
DATA SAMPLING FREQUENCY: 0.200 Hz



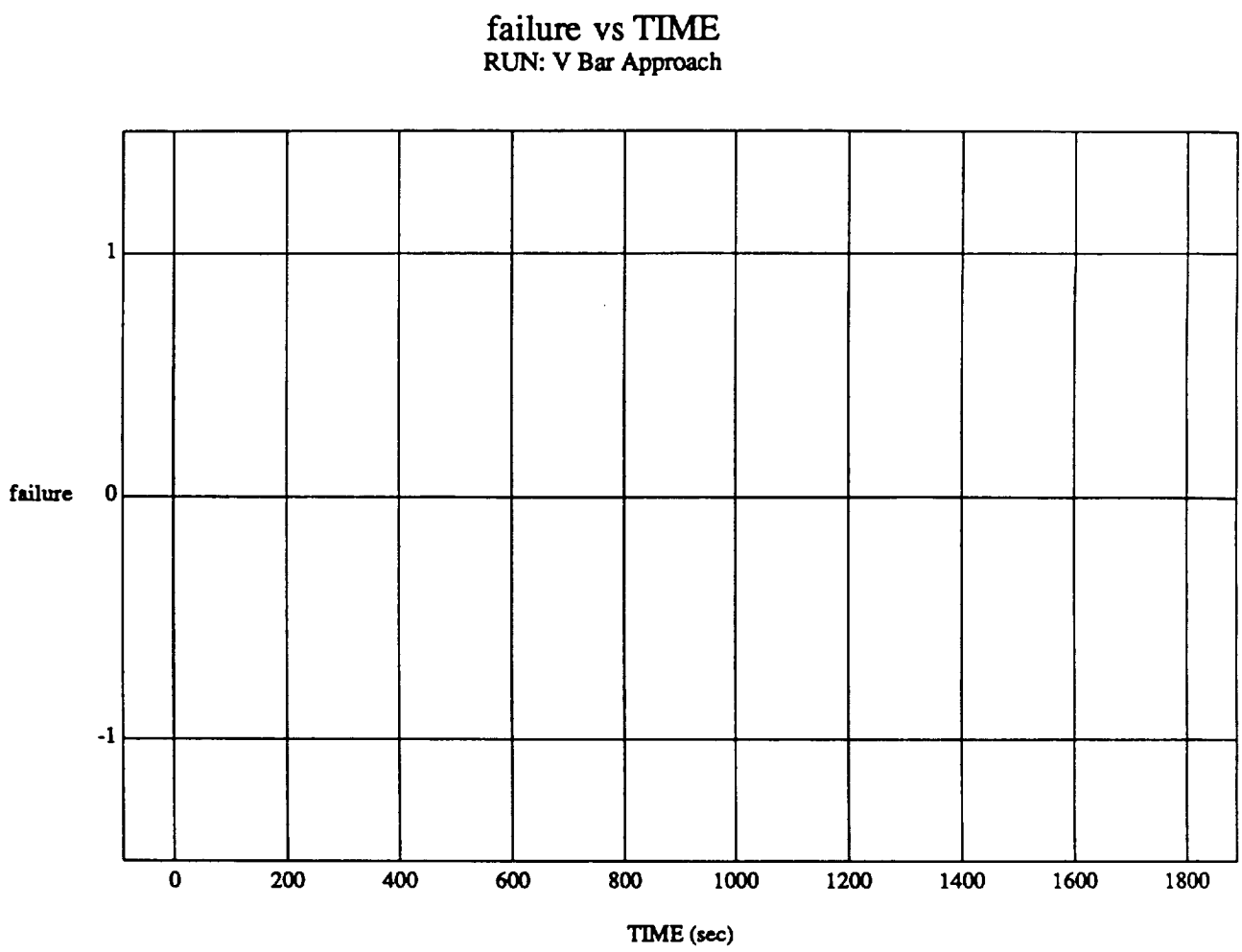
MODULE: ORBITER.lm_elev
DATA SAMPLING FREQUENCY: 0.200 Hz

unusualness vs TIME

RUN: V Bar Approach

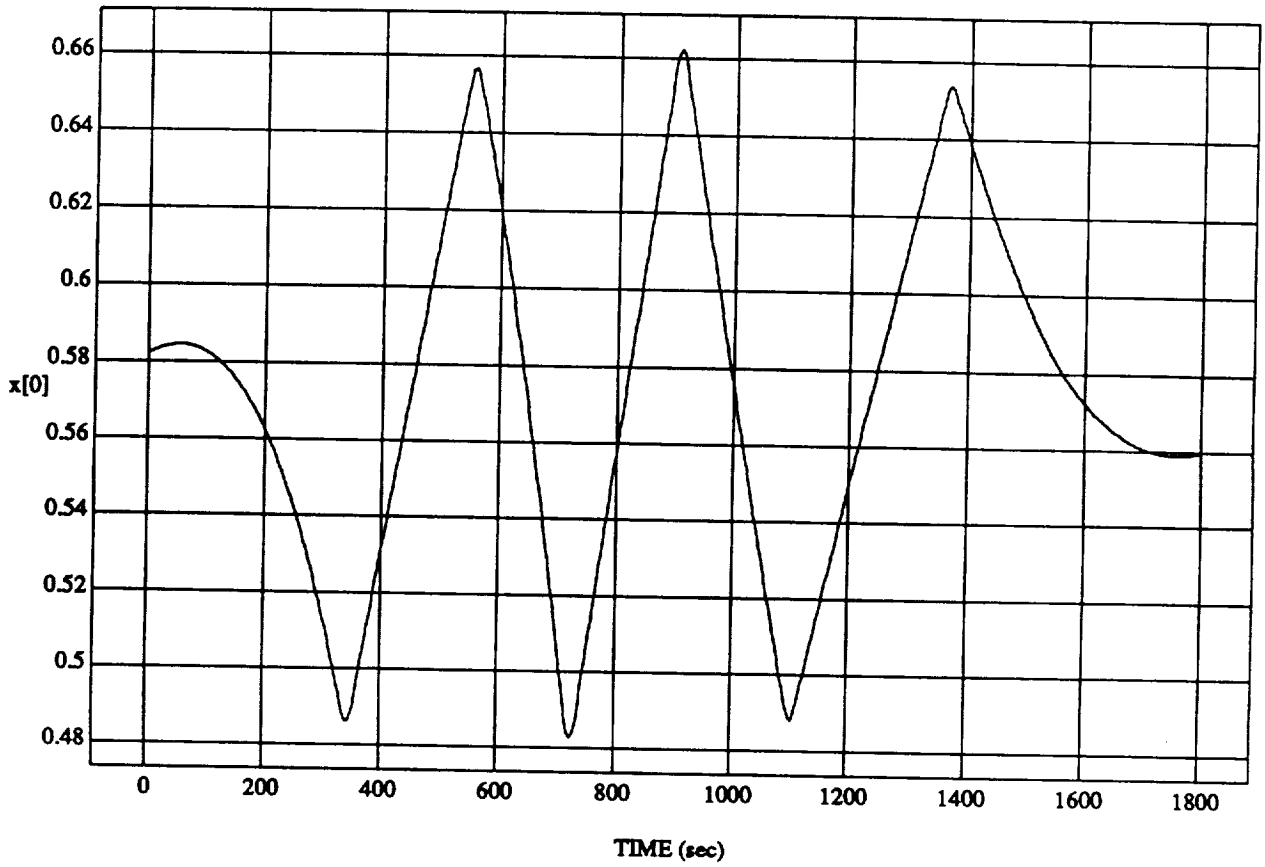


MODULE: ORBITER.im_elev
DATA SAMPLING FREQUENCY: 0.200 Hz

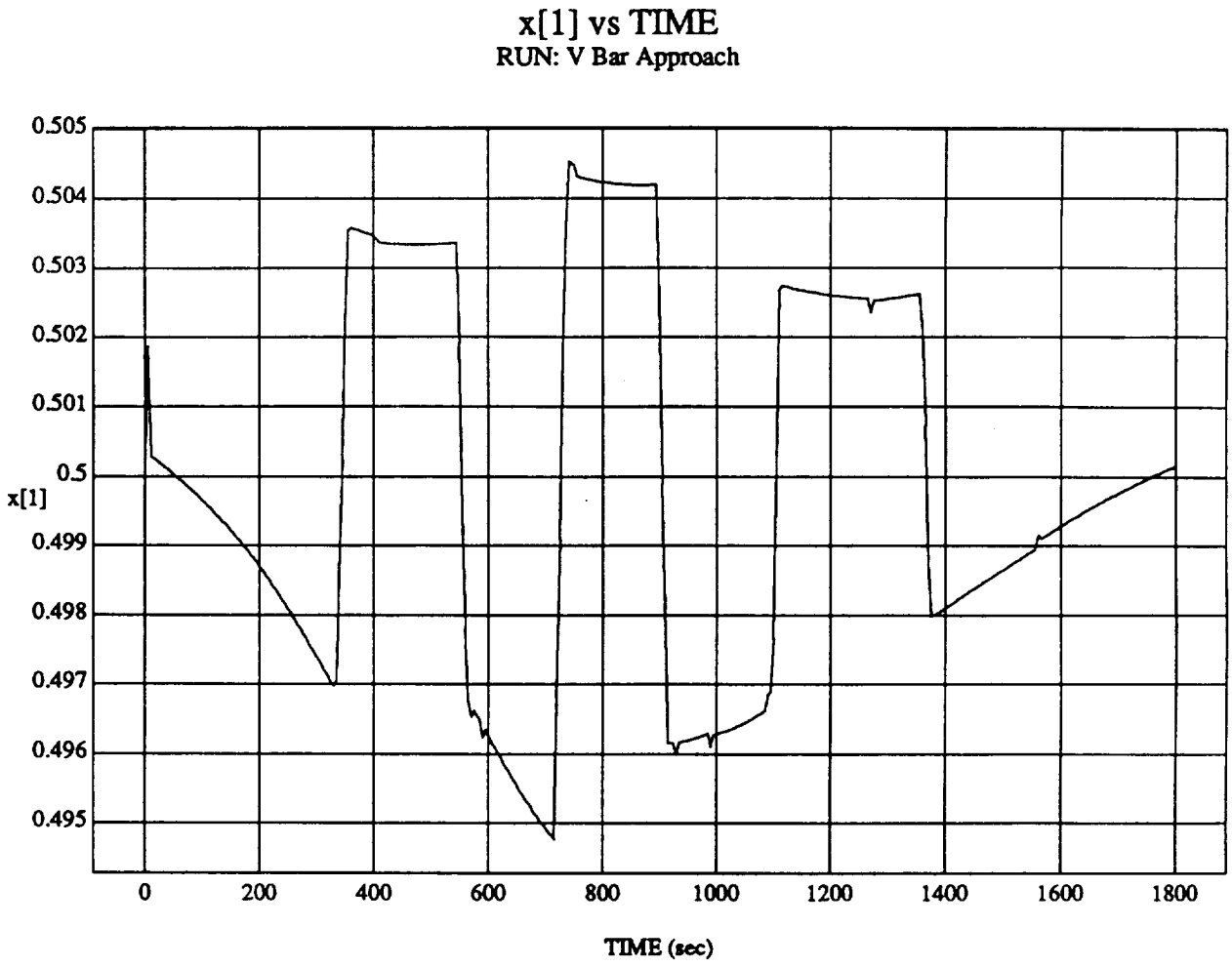


MODULE: ORBITER.lm_elev
DATA SAMPLING FREQUENCY: 0.200 Hz

x[0] vs TIME
RUN: V Bar Approach



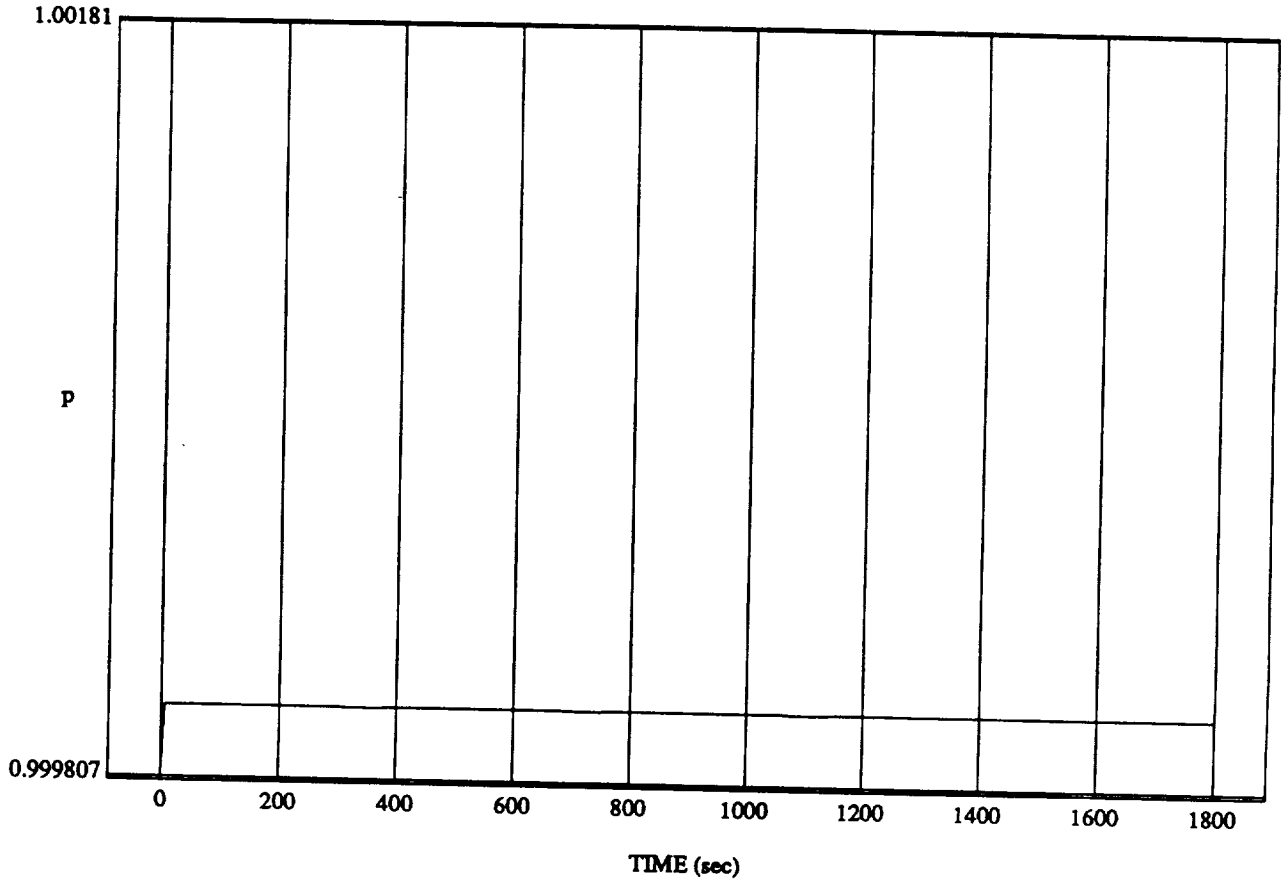
MODULE: ORBITER.lm_elev
DATA SAMPLING FREQUENCY: 0.200 Hz



MODULE: ORBITER.lm_elev
DATA SAMPLING FREQUENCY: 0.200 Hz

SIMULATION APPLICATION: ARIC Translational Controller Simulation

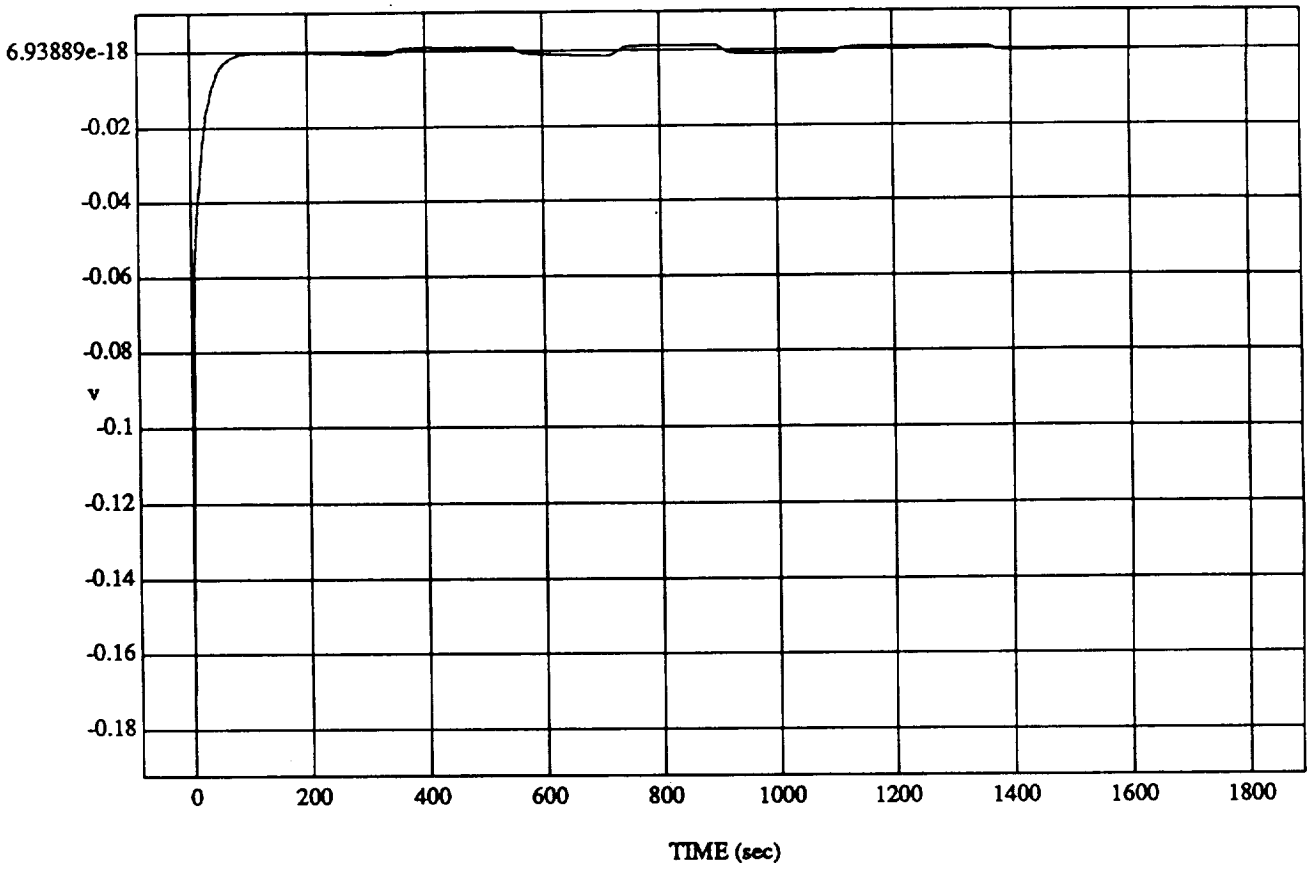
p vs TIME
RUN: V Bar Approach



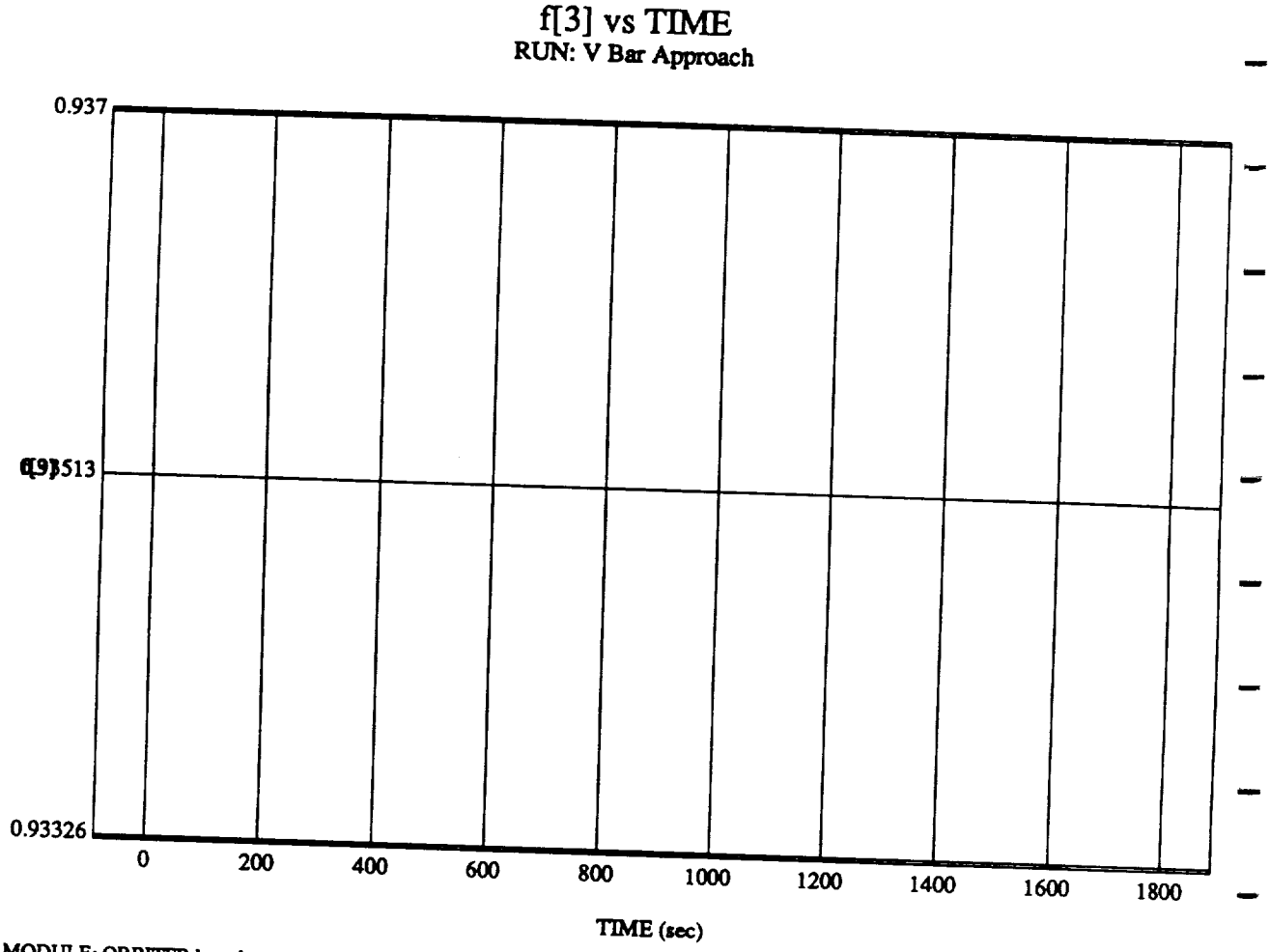
MODULE: ORBITER.lm_elev
DATA SAMPLING FREQUENCY: 0.200 Hz

SIMULATION APPLICATION: ARIC Translational Controller Simulation

v vs TIME
RUN: V Bar Approach

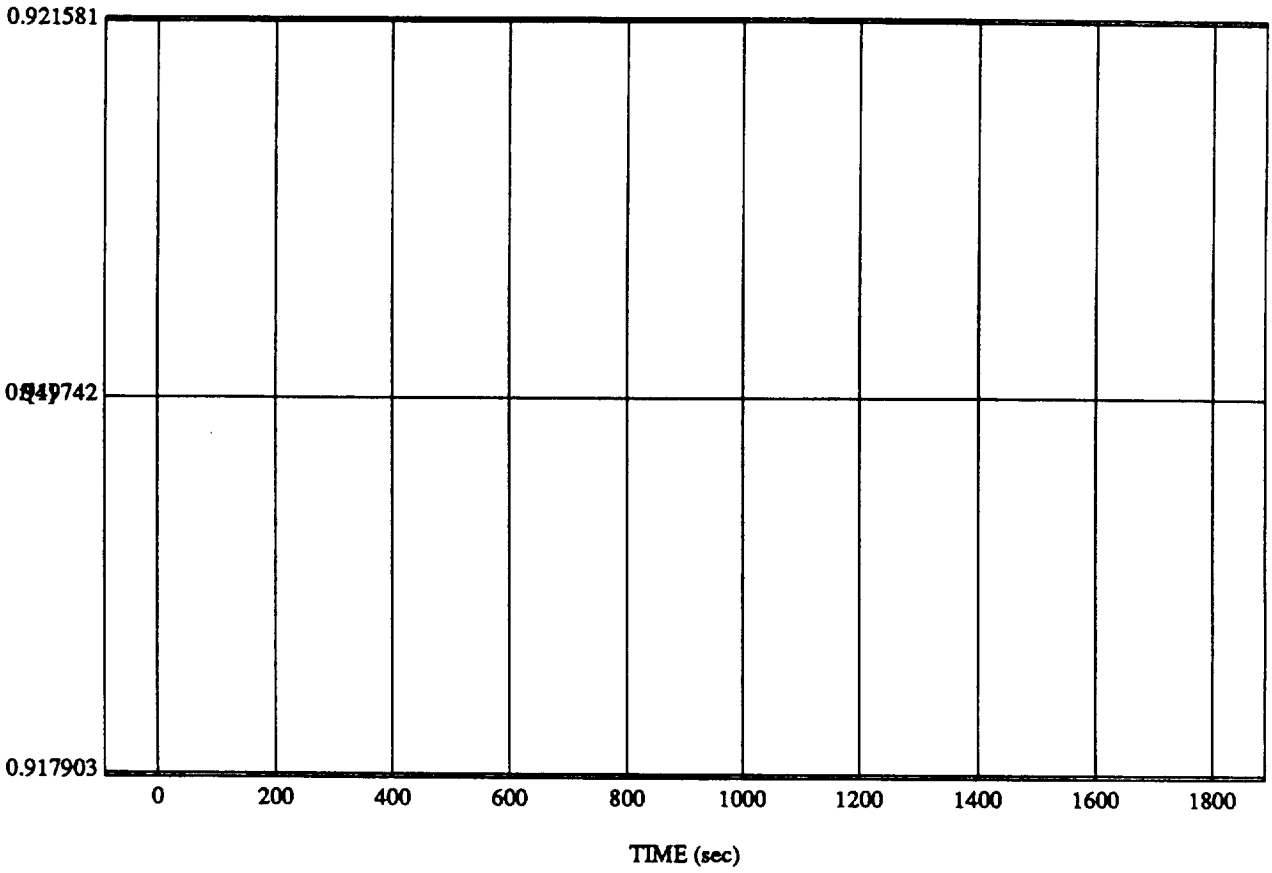


MODULE: ORBITER.lm_elev
DATA SAMPLING FREQUENCY: 0.200 Hz



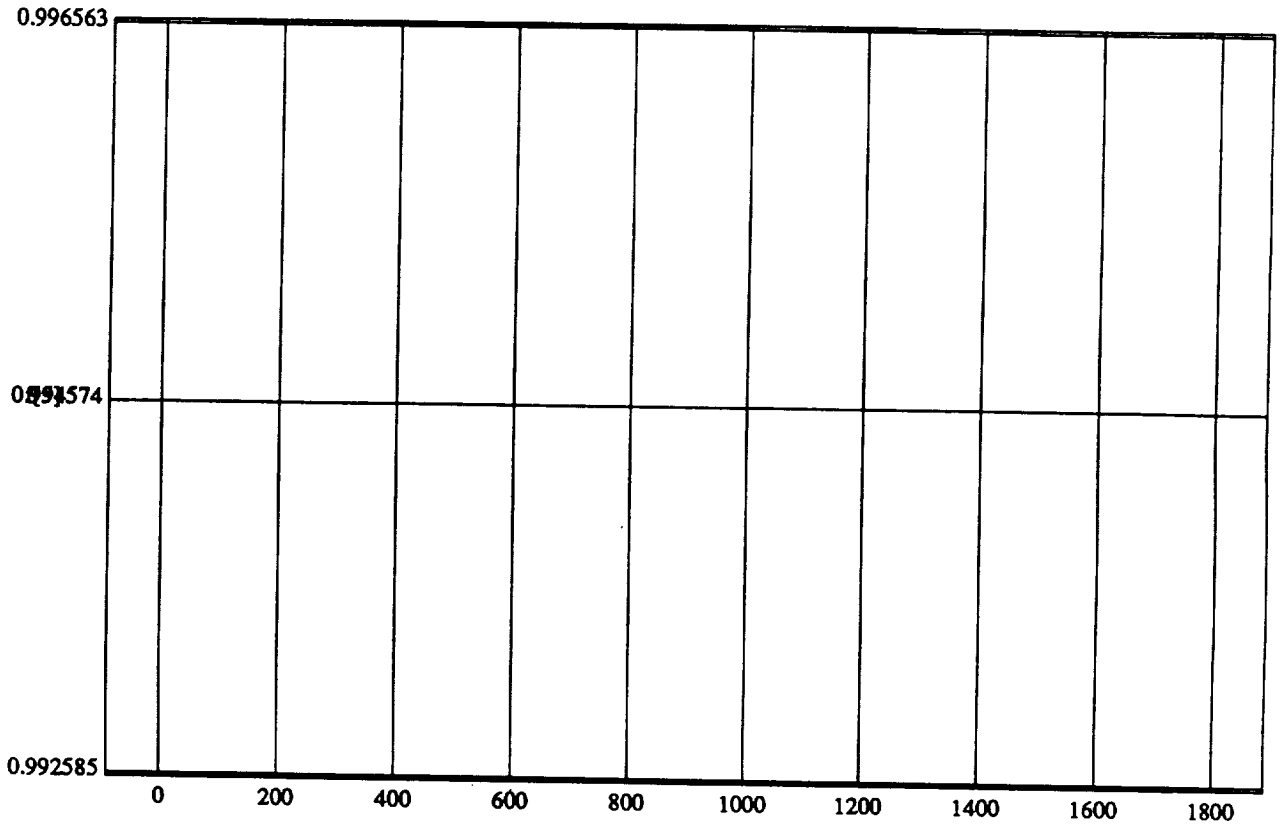
MODULE: ORBITER.lm_elev
DATA SAMPLING FREQUENCY: 0.200 Hz

f[4] vs TIME
RUN: V Bar Approach



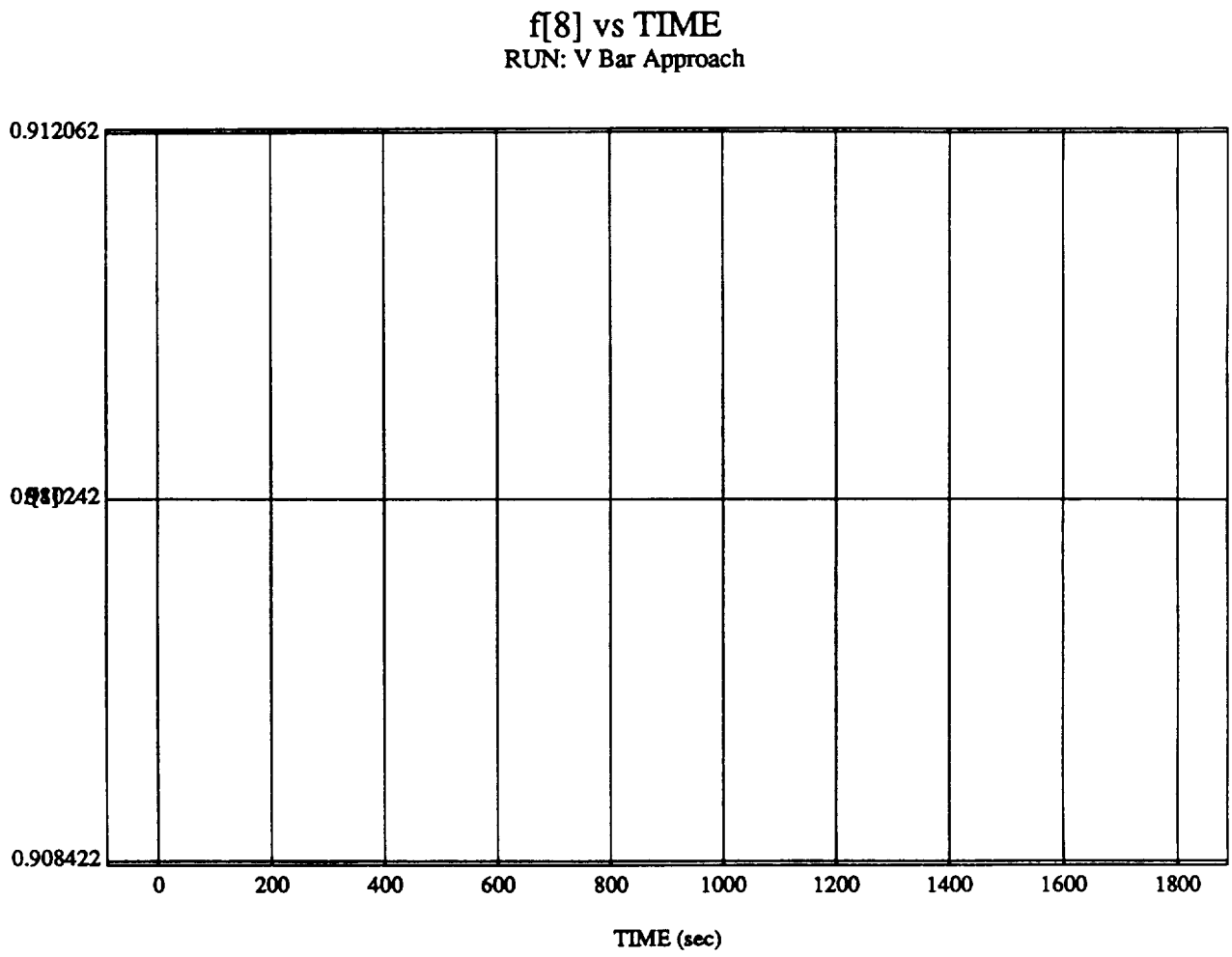
MODULE: ORBITER.lm_elev
DATA SAMPLING FREQUENCY: 0.200 Hz

f[5] vs TIME
RUN: V Bar Approach



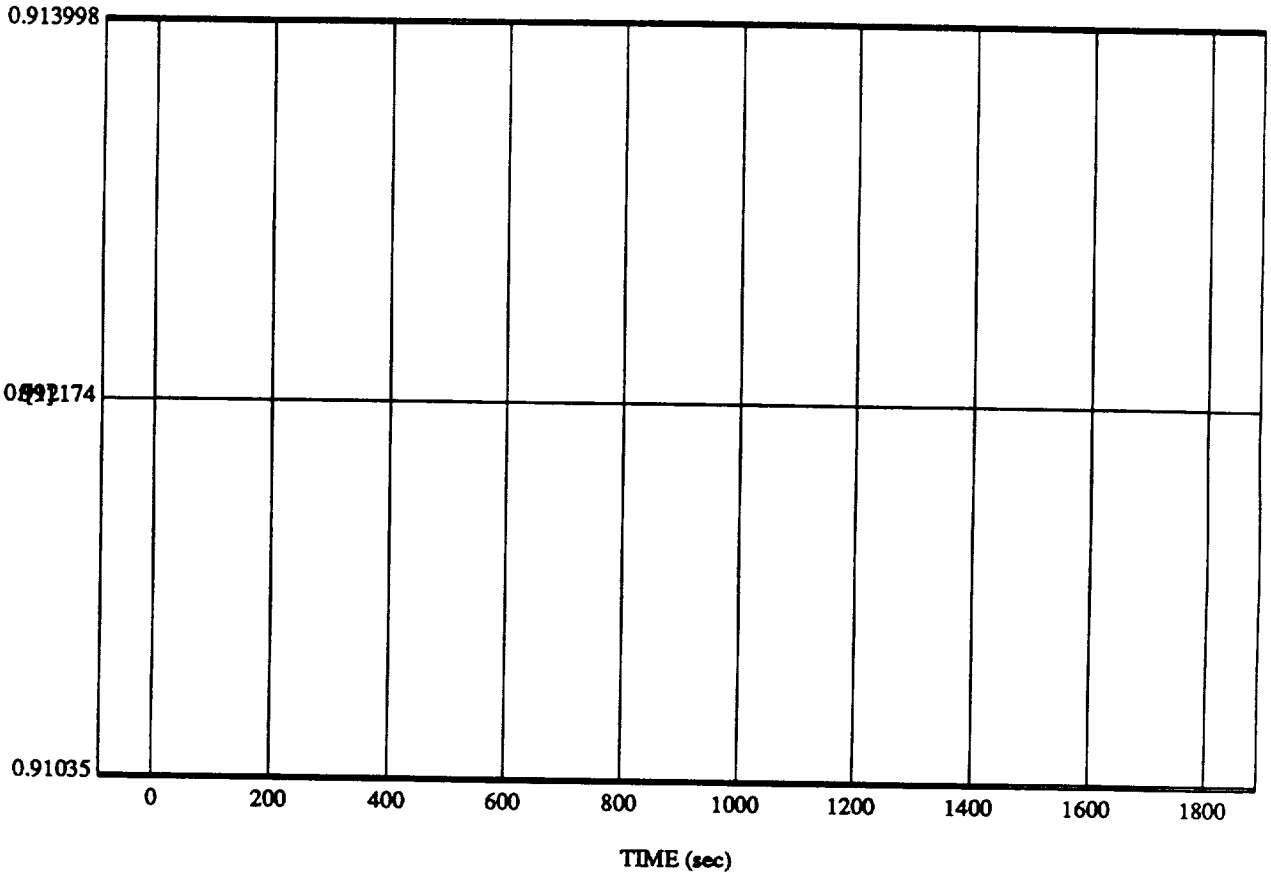
MODULE: ORBITER.lm_elev
DATA SAMPLING FREQUENCY: 0.200 Hz

SIMULATION APPLICATION: ARIC Translational Controller Simulation

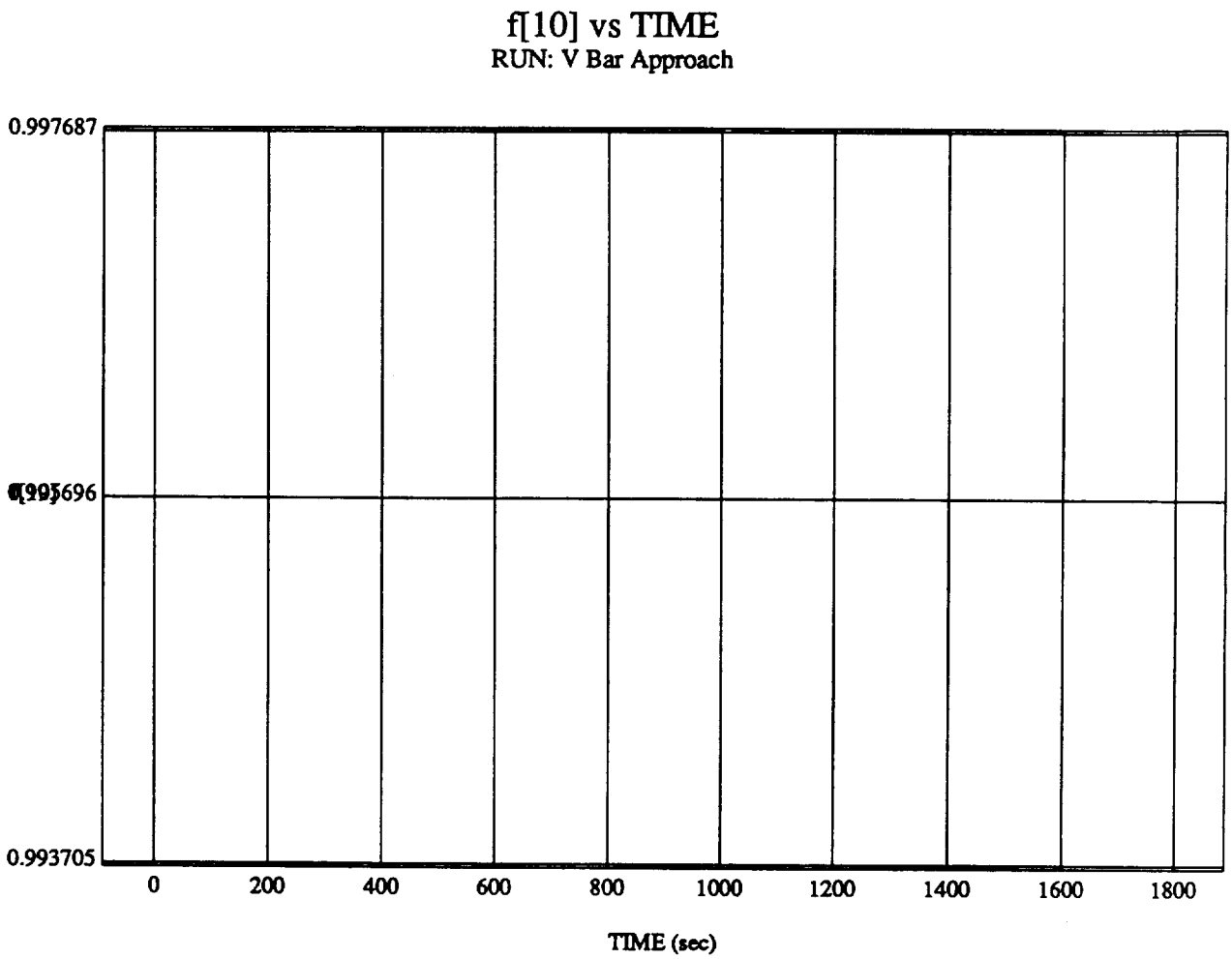


MODULE: ORBITER.lm_elev
DATA SAMPLING FREQUENCY: 0.200 Hz

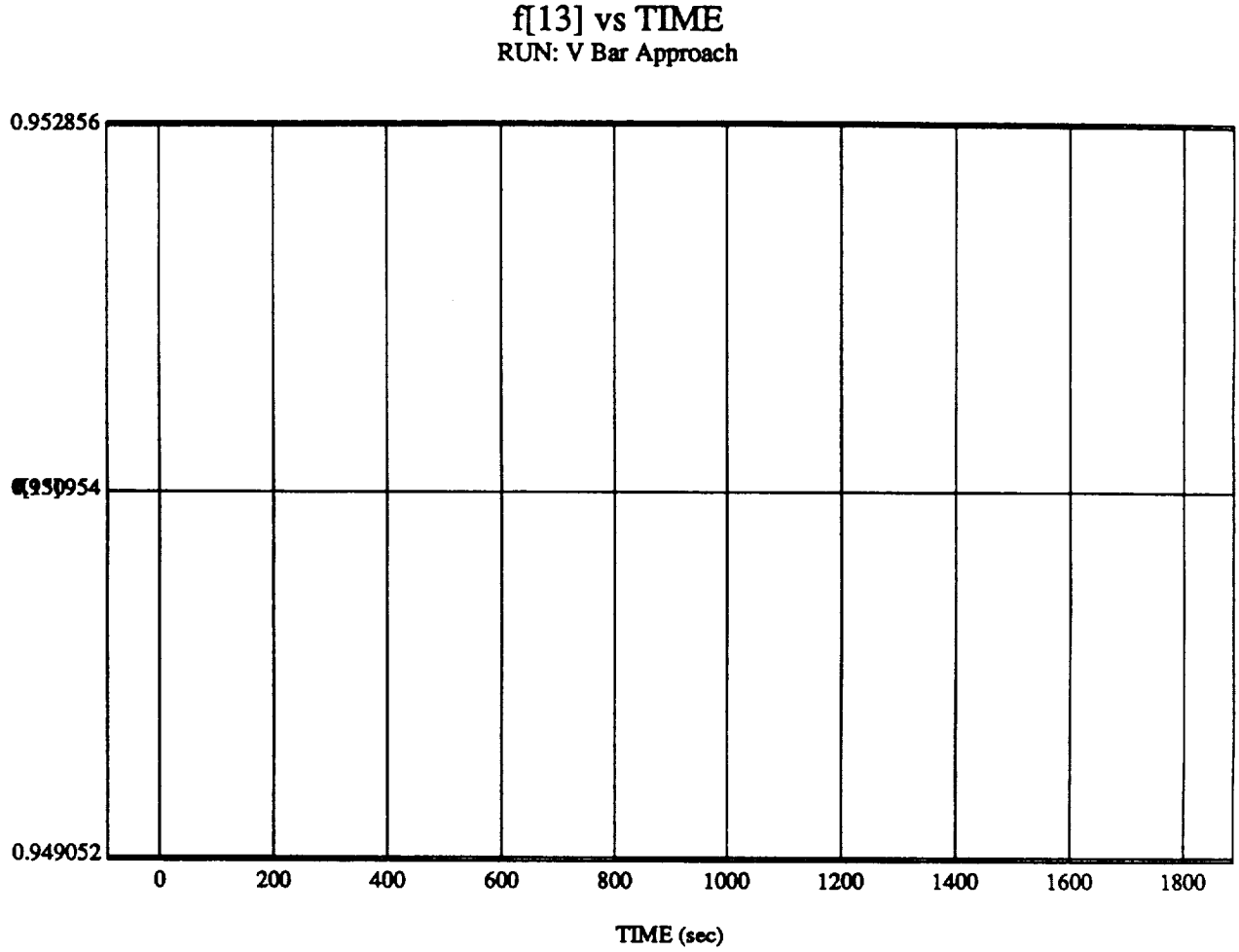
f[9] vs TIME
RUN: V Bar Approach



MODULE: ORBITER.lm_elev
DATA SAMPLING FREQUENCY: 0.200 Hz

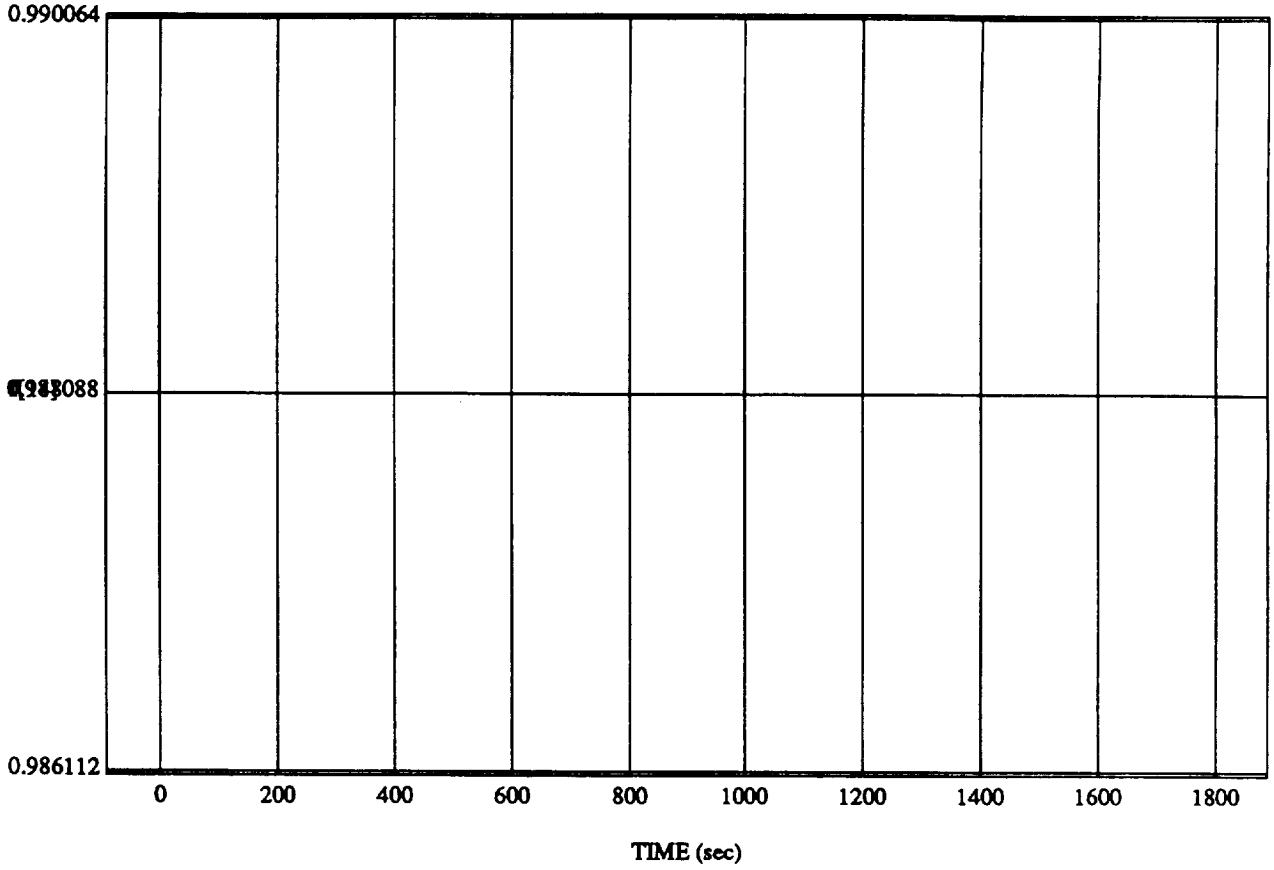


MODULE: ORBITER.lm_elev
DATA SAMPLING FREQUENCY: 0.200 Hz



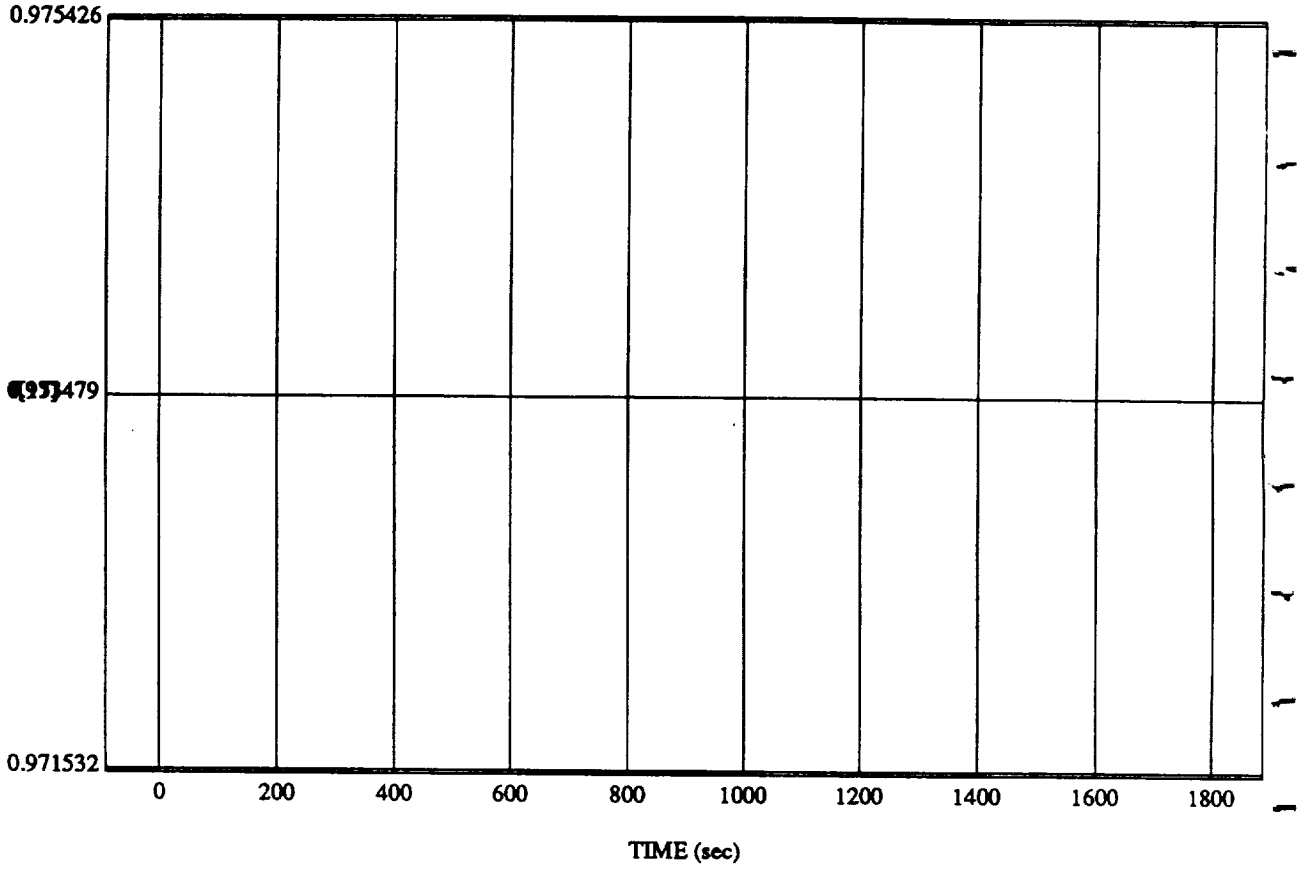
MODULE: ORBITER.lm_elev
DATA SAMPLING FREQUENCY: 0.200 Hz

f[14] vs TIME
RUN: V Bar Approach

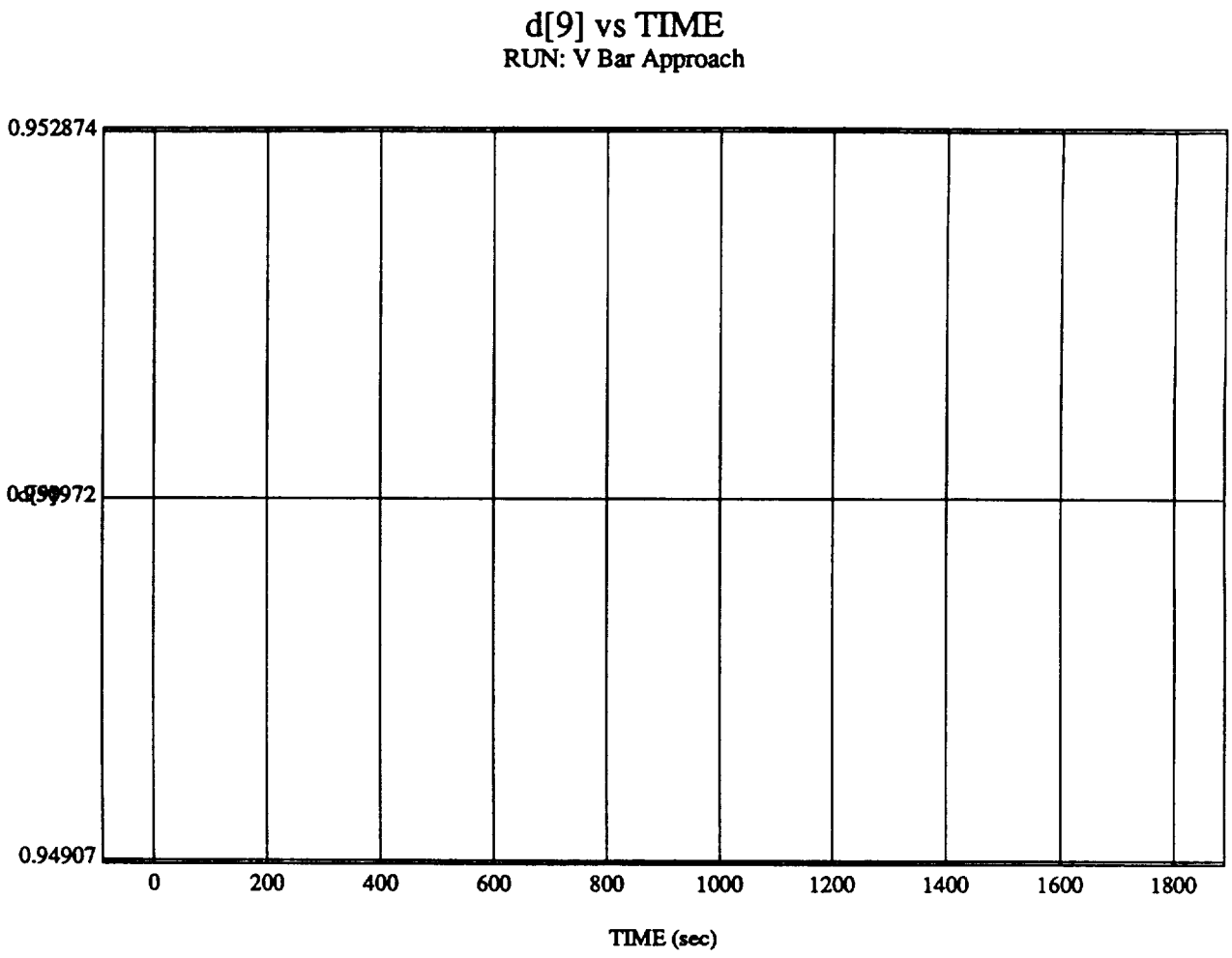


MODULE: ORBITER.lm_elev
DATA SAMPLING FREQUENCY: 0.200 Hz

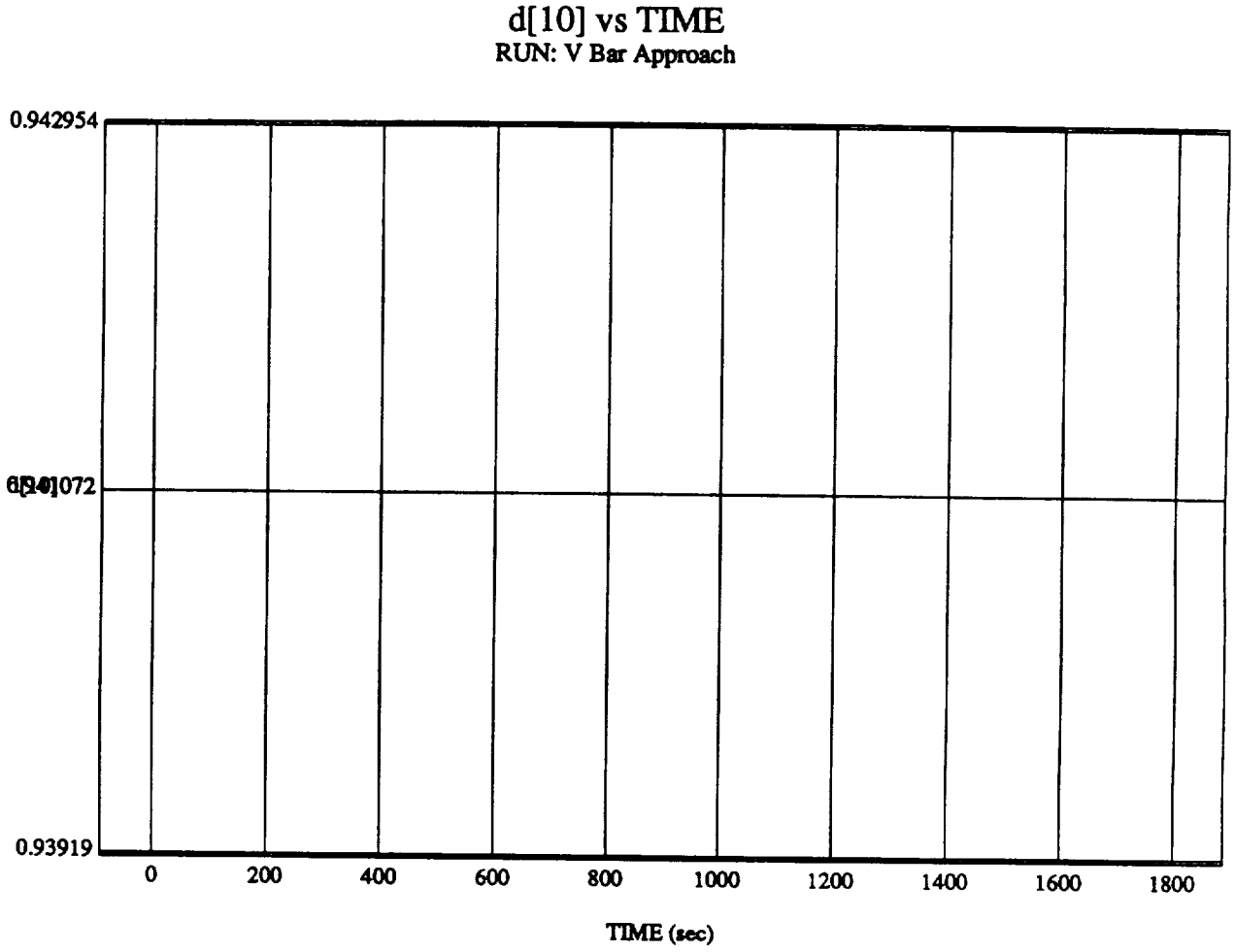
f[15] vs TIME
RUN: V Bar Approach



MODULE: ORBITER.lm_elev
DATA SAMPLING FREQUENCY: 0.200 Hz

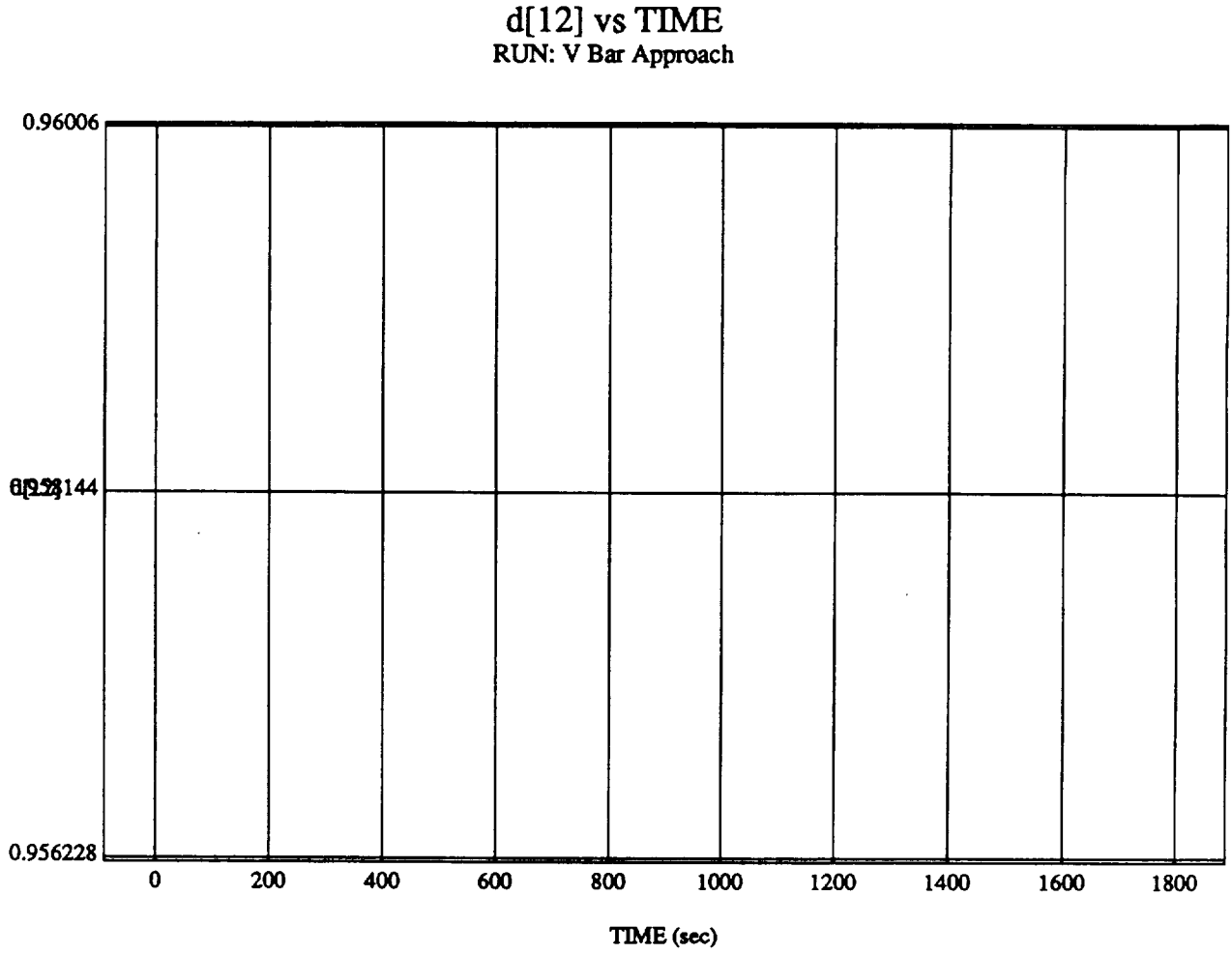


MODULE: ORBITER.lm_elev
DATA SAMPLING FREQUENCY: 0.200 Hz

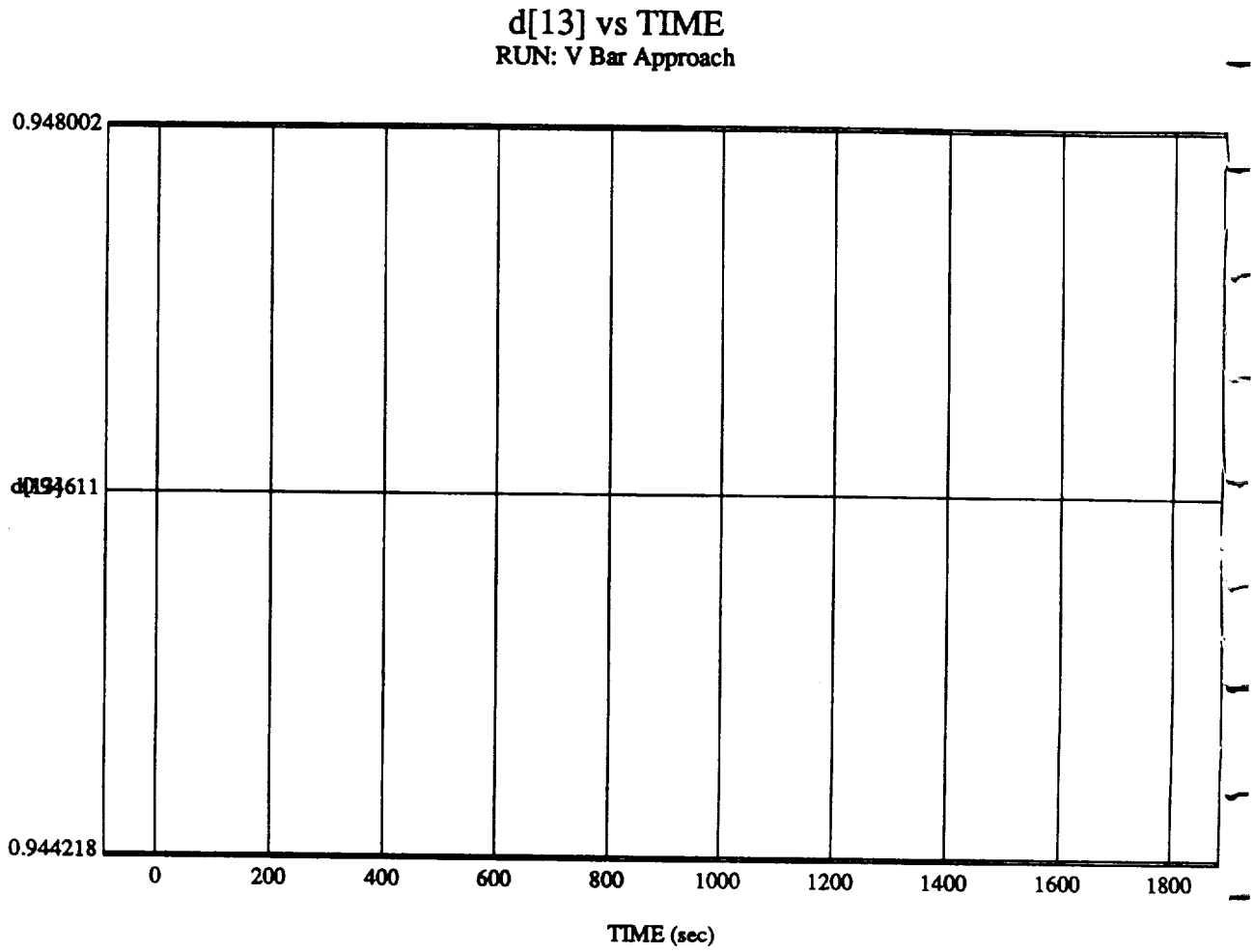


MODULE: ORBITER.lm_elev
DATA SAMPLING FREQUENCY: 0.200 Hz

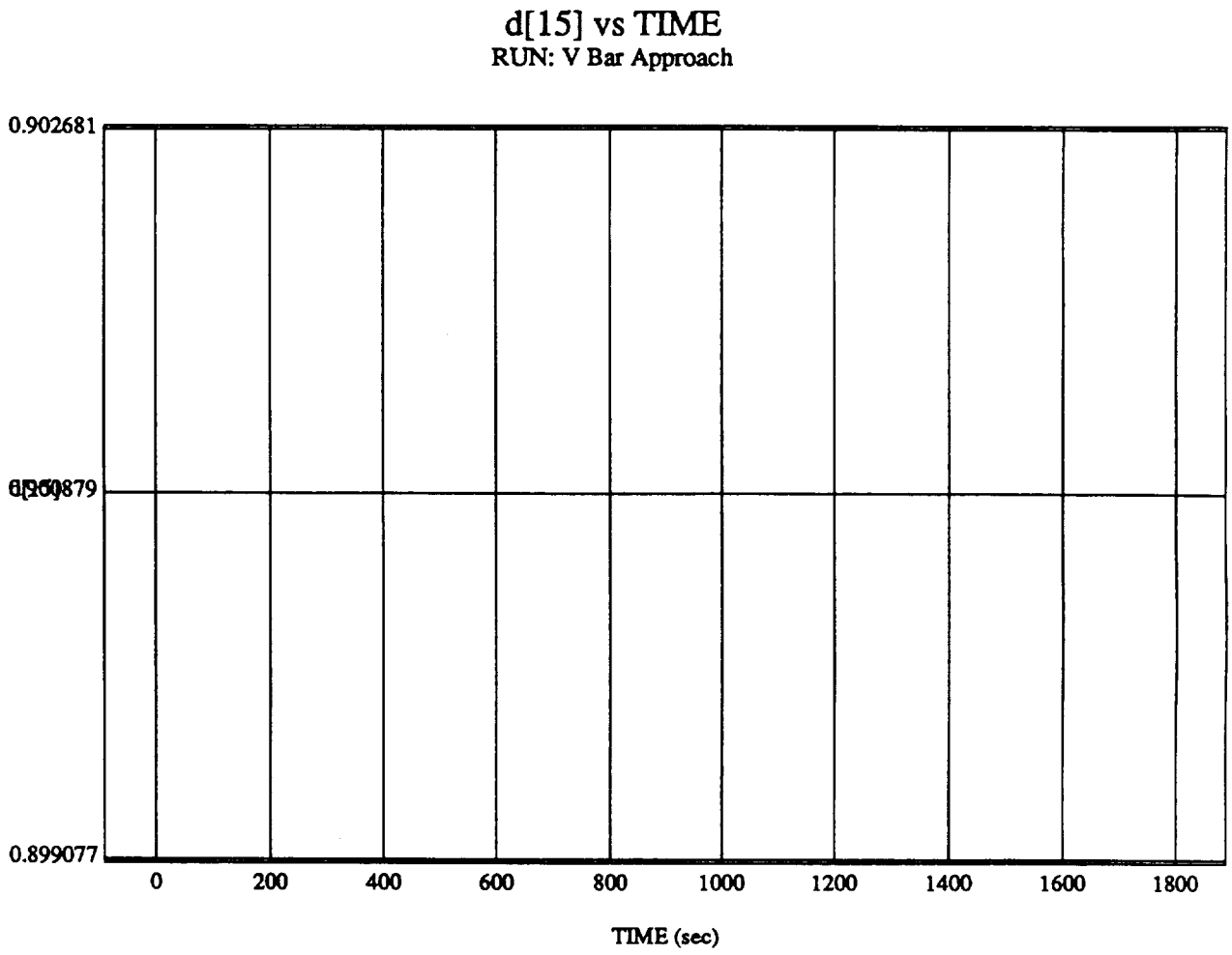
SIMULATION APPLICATION: ARIC Translational Controller Simulation



MODULE: ORBITER.lm_elev
DATA SAMPLING FREQUENCY: 0.200 Hz

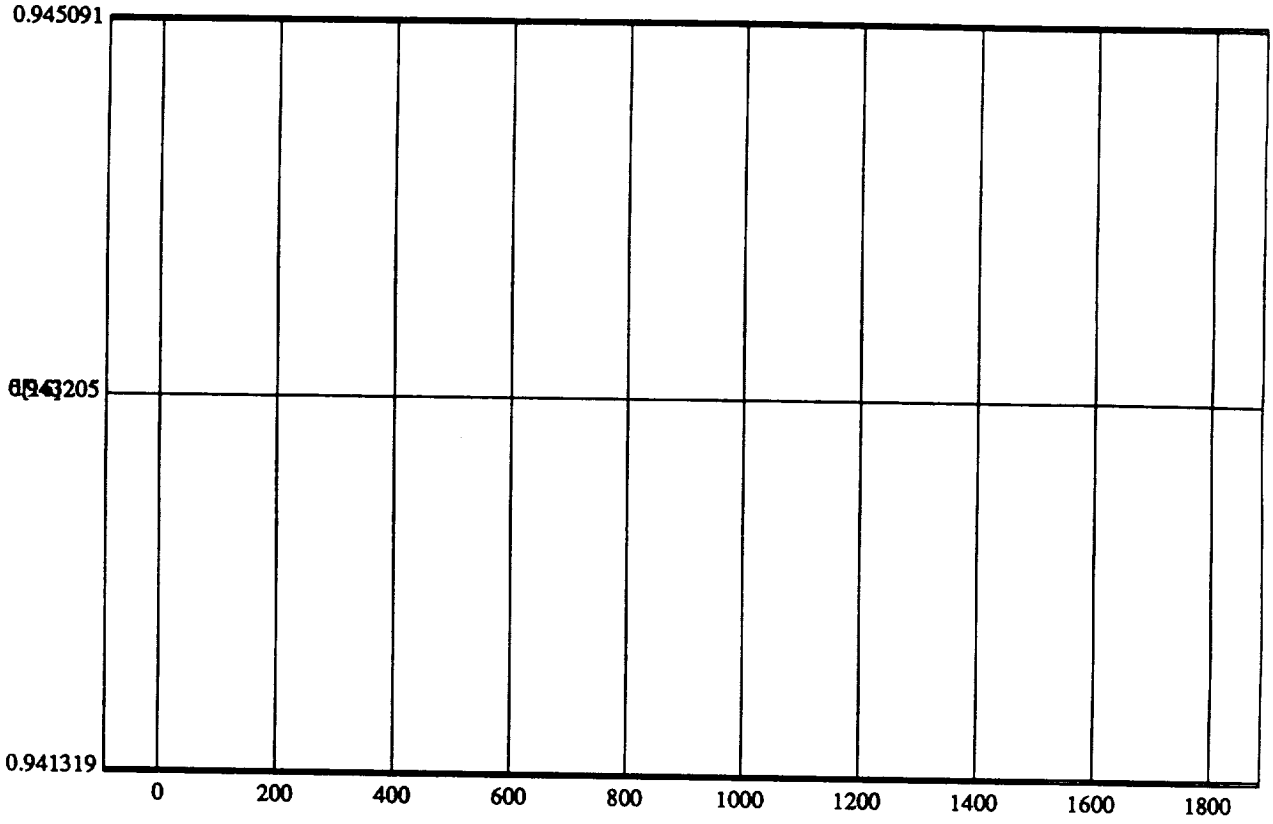


MODULE: ORBITER.lm_elev
DATA SAMPLING FREQUENCY: 0.200 Hz

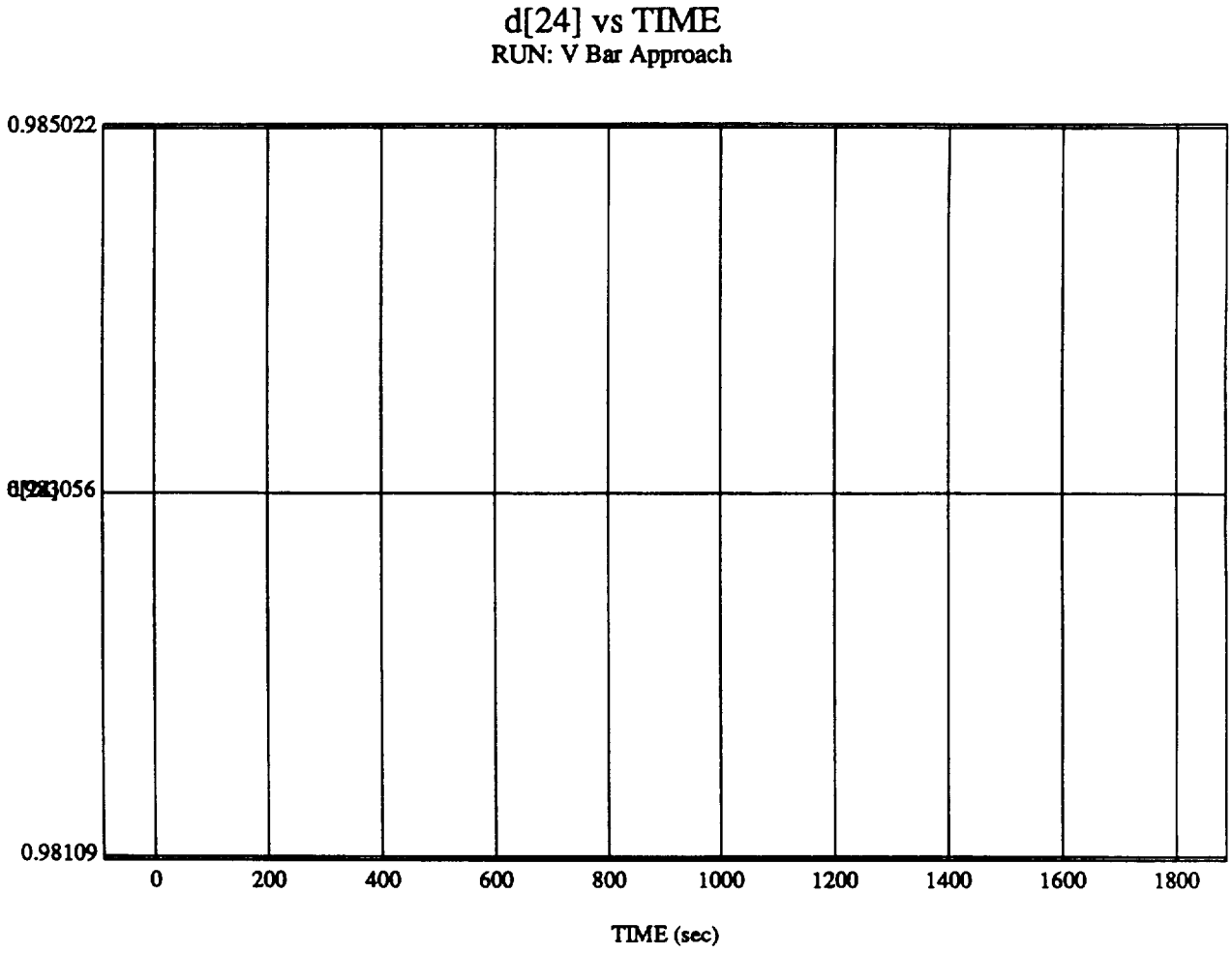


MODULE: ORBITER.lm_elev
DATA SAMPLING FREQUENCY: 0.200 Hz

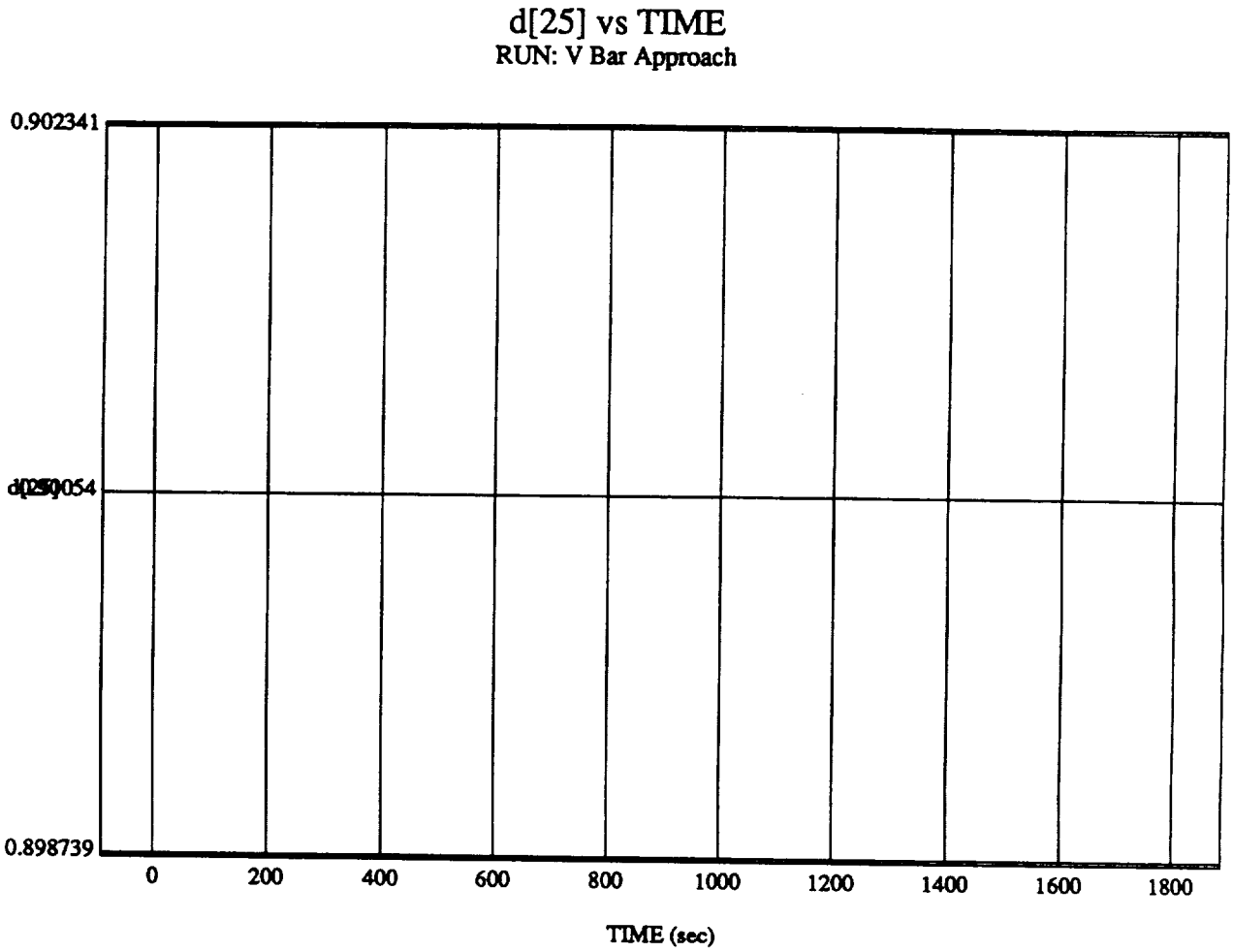
d[16] vs TIME
RUN: V Bar Approach



MODULE: ORBITER.lm_elev
DATA SAMPLING FREQUENCY: 0.200 Hz

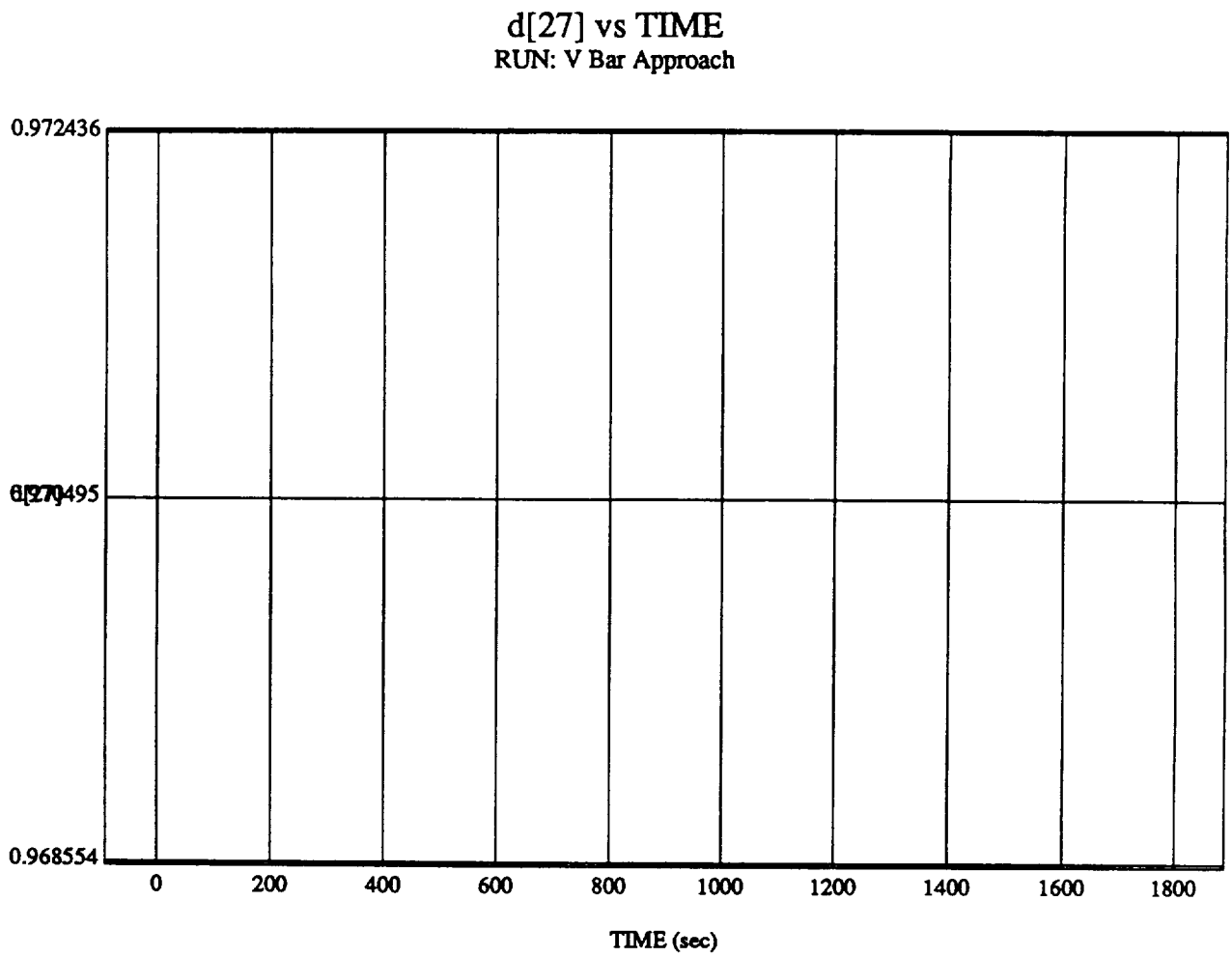


MODULE: ORBITER.lm_elev
DATA SAMPLING FREQUENCY: 0.200 Hz

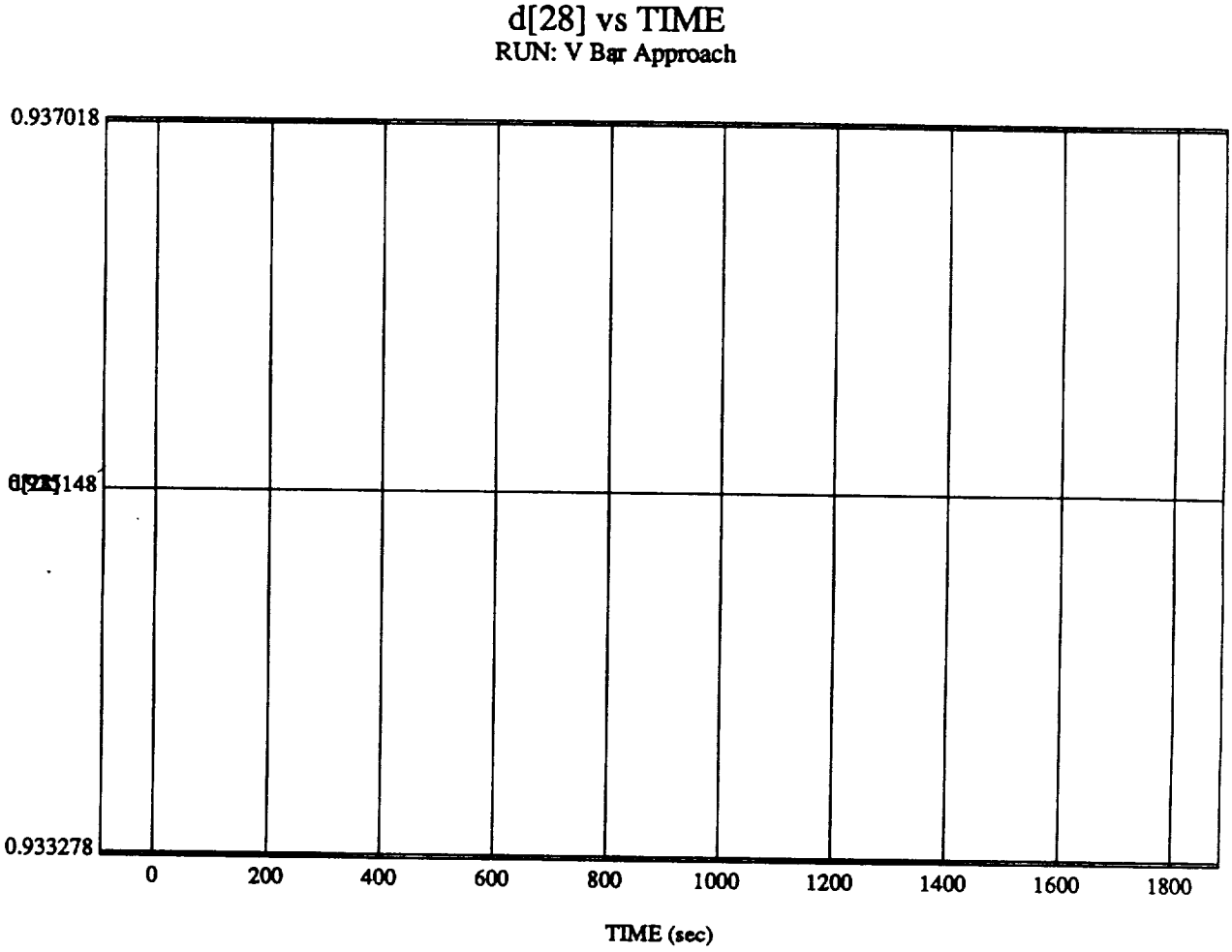


MODULE: ORBITER.lm_elev
DATA SAMPLING FREQUENCY: 0.200 Hz

SIMULATION APPLICATION: ARIC Translational Controller Simulation

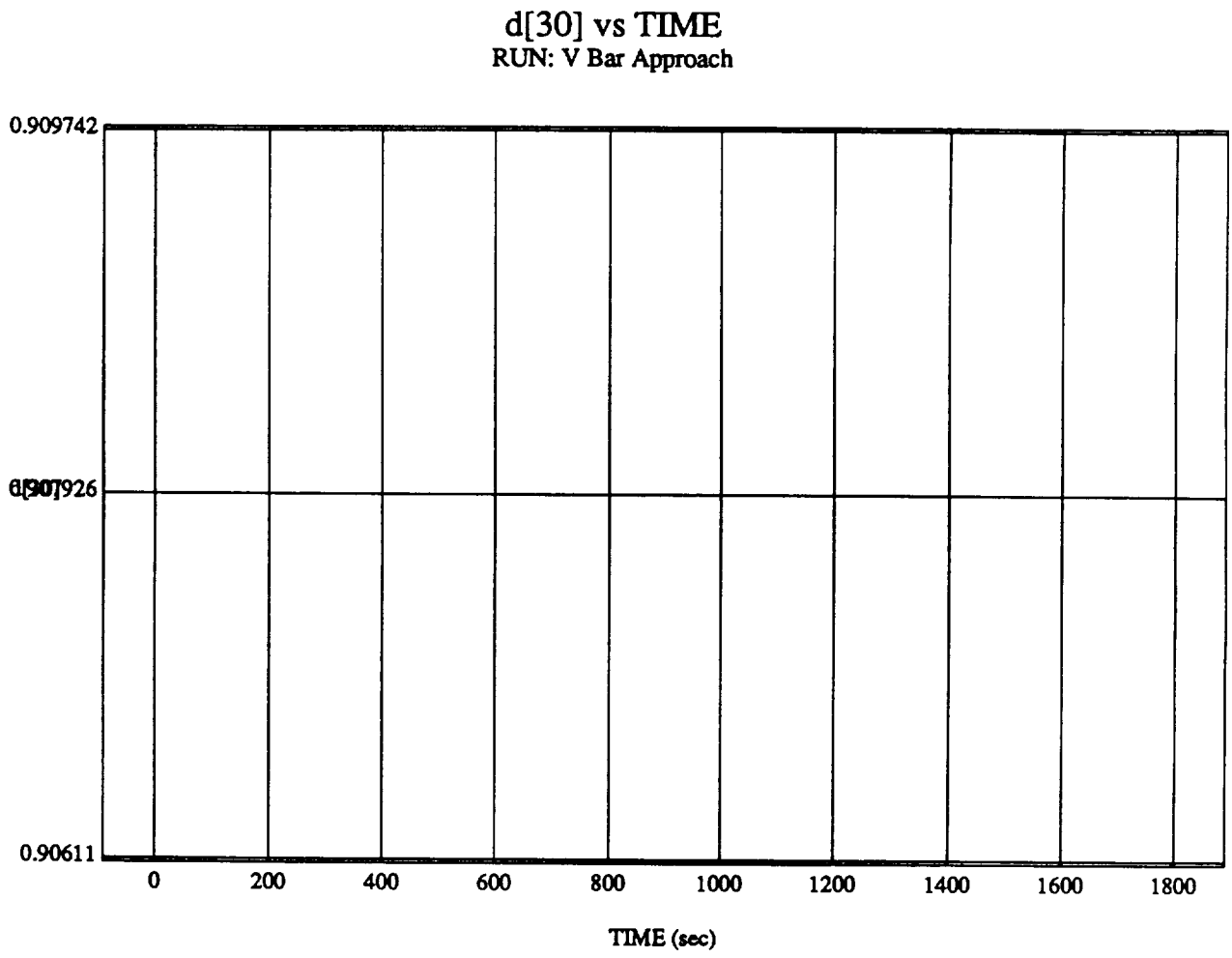


MODULE: ORBITER.lm_elev
DATA SAMPLING FREQUENCY: 0.200 Hz



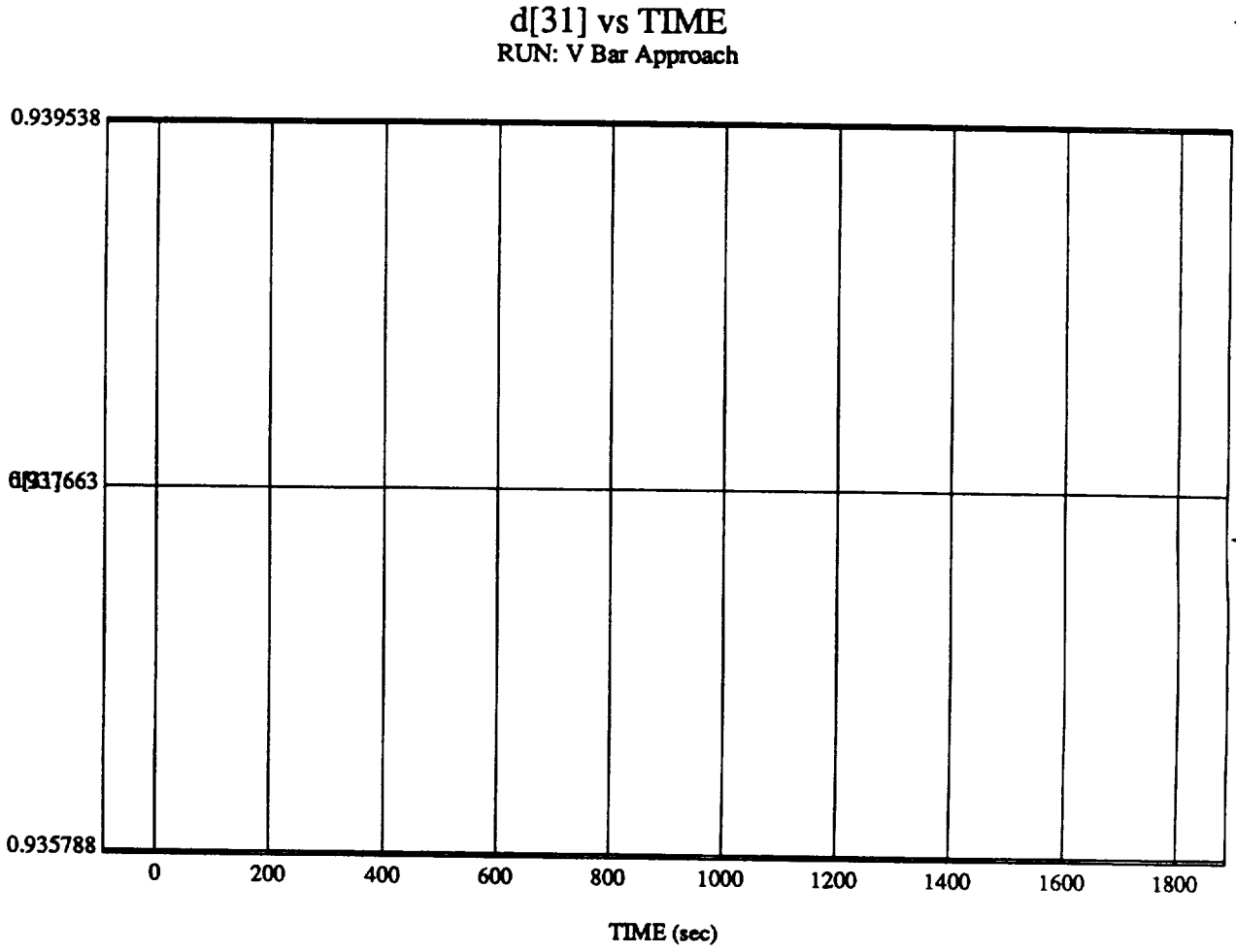
MODULE: ORBITER.lm_elev
DATA SAMPLING FREQUENCY: 0.200 Hz

SIMULATION APPLICATION: ARIC Translational Controller Simulation

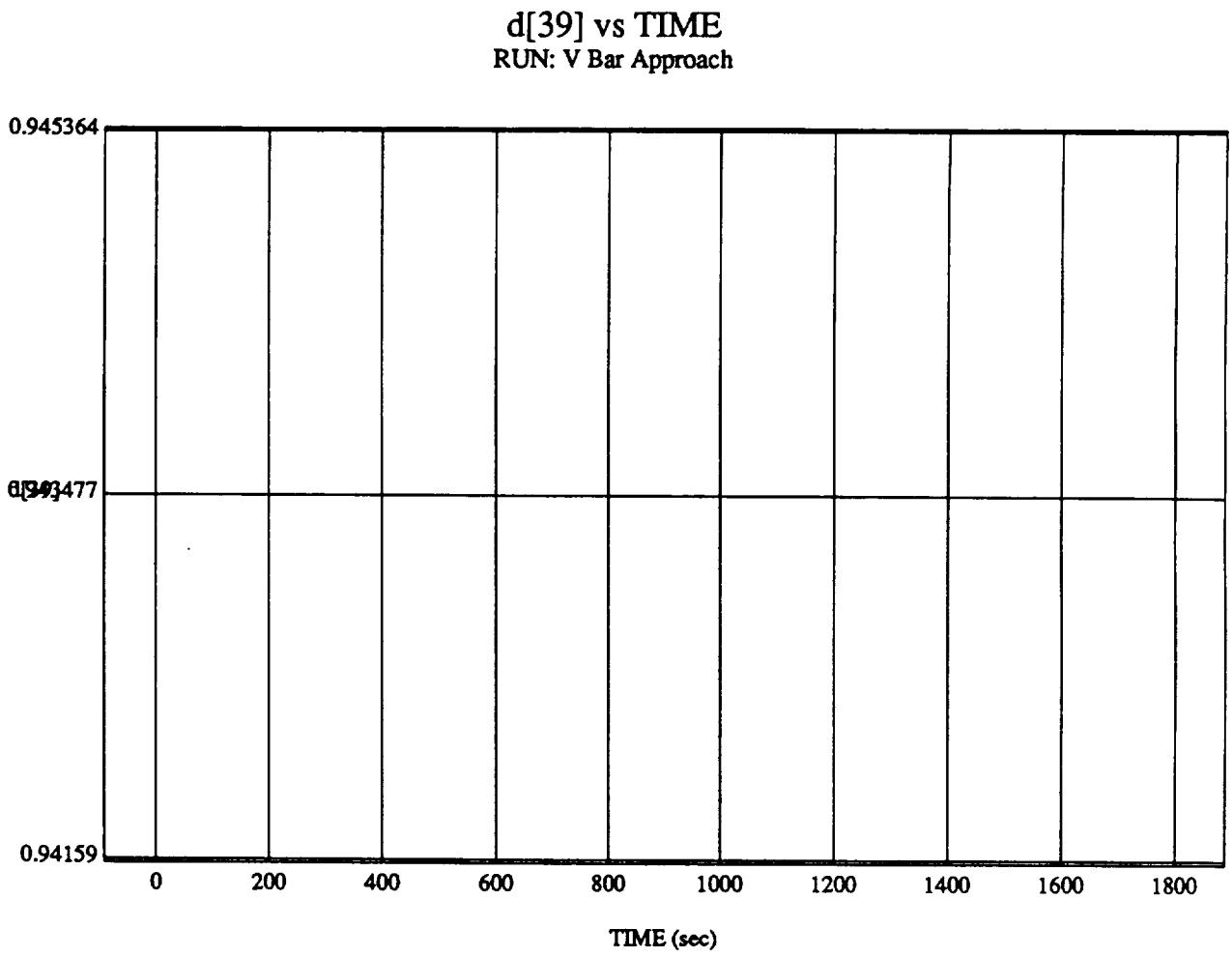


MODULE: ORBITER.lm_elev
DATA SAMPLING FREQUENCY: 0.200 Hz

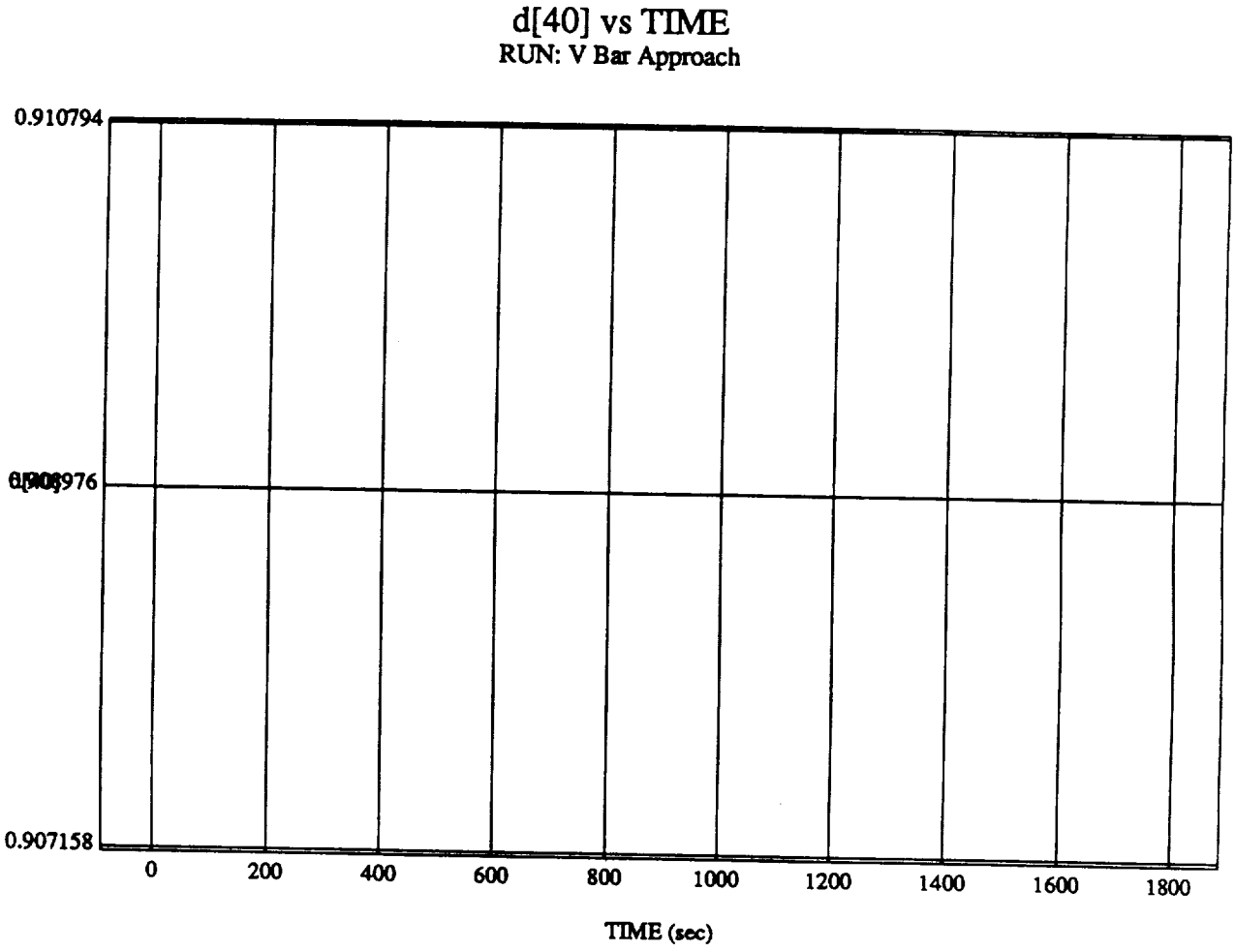
SIMULATION APPLICATION: ARIC Translational Controller Simulation



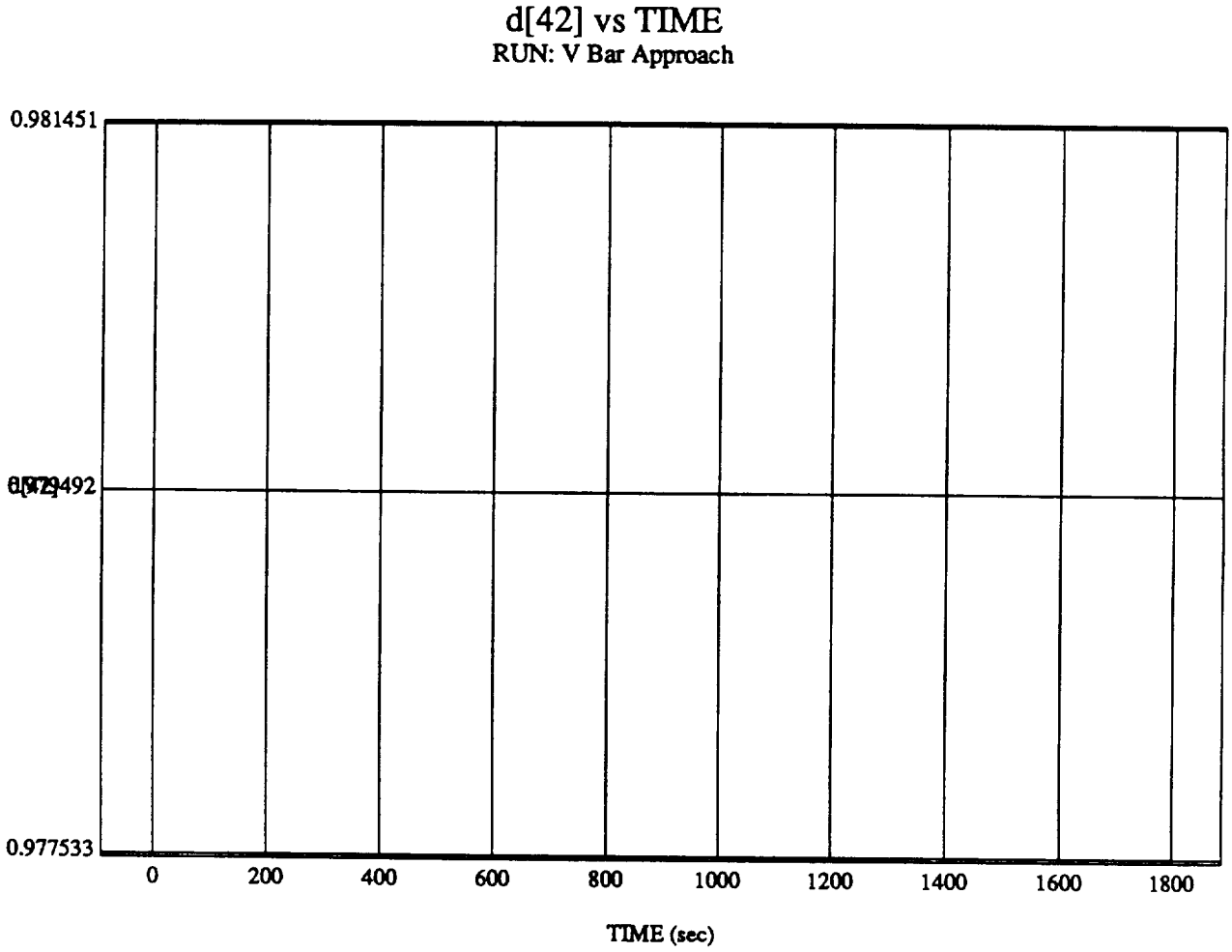
MODULE: ORBITER.lm_elev
DATA SAMPLING FREQUENCY: 0.200 Hz



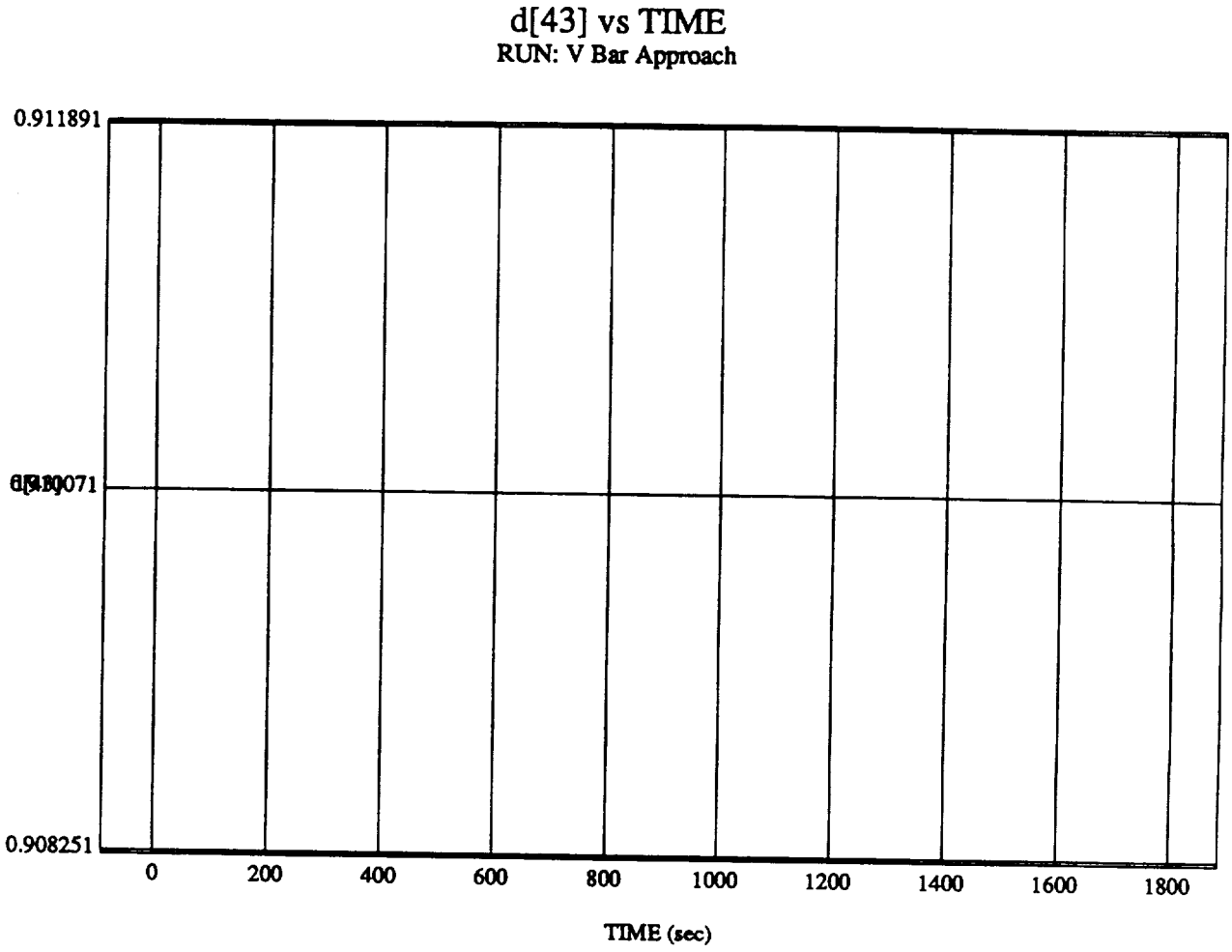
MODULE: ORBITER.lm_elev
DATA SAMPLING FREQUENCY: 0.200 Hz



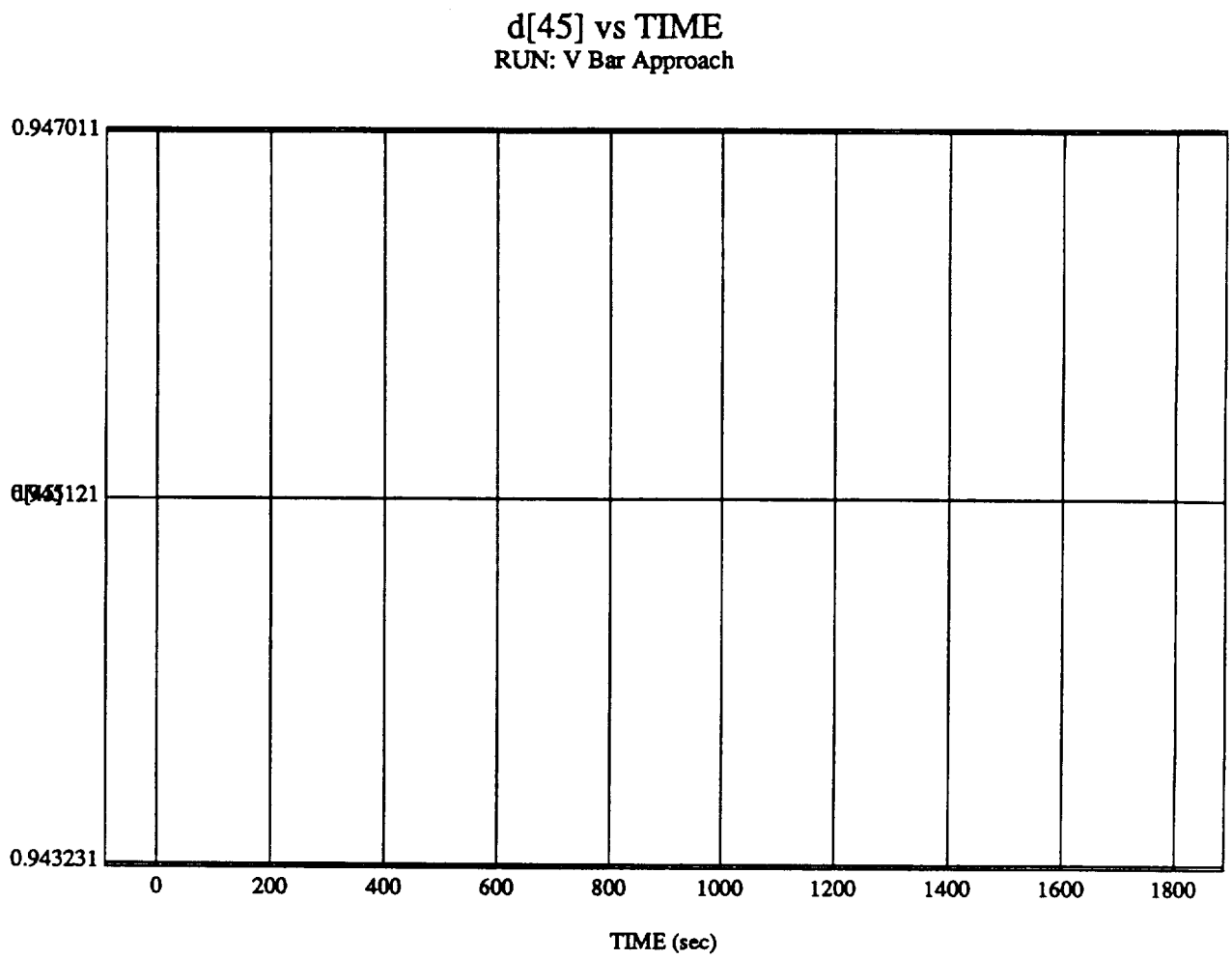
MODULE: ORBITER.lm_elev
DATA SAMPLING FREQUENCY: 0.200 Hz



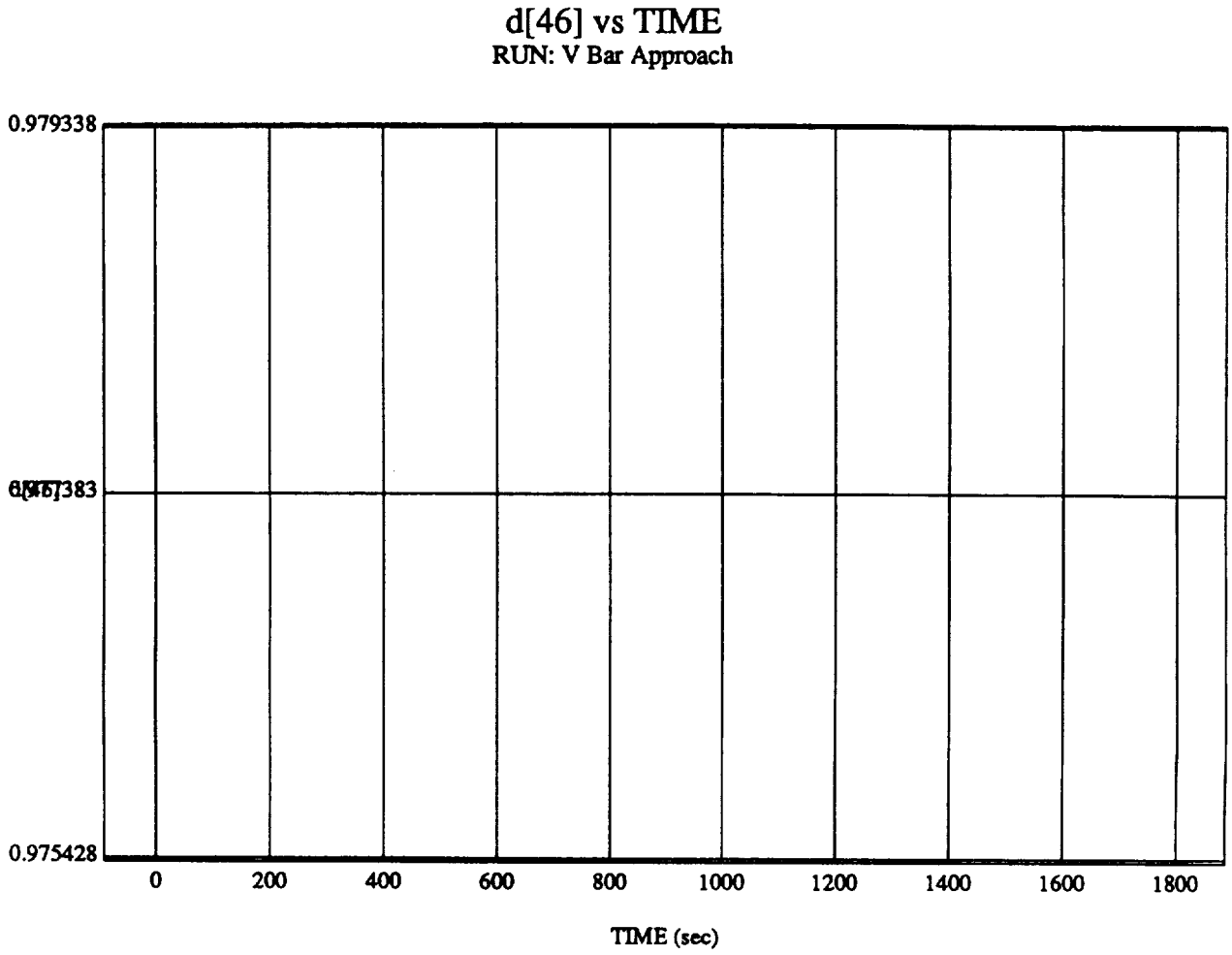
MODULE: ORBITER.lm_elev
DATA SAMPLING FREQUENCY: 0.200 Hz



MODULE: ORBITER.lm_elev
DATA SAMPLING FREQUENCY: 0.200 Hz



MODULE: ORBITER.lm_elev
DATA SAMPLING FREQUENCY: 0.200 Hz

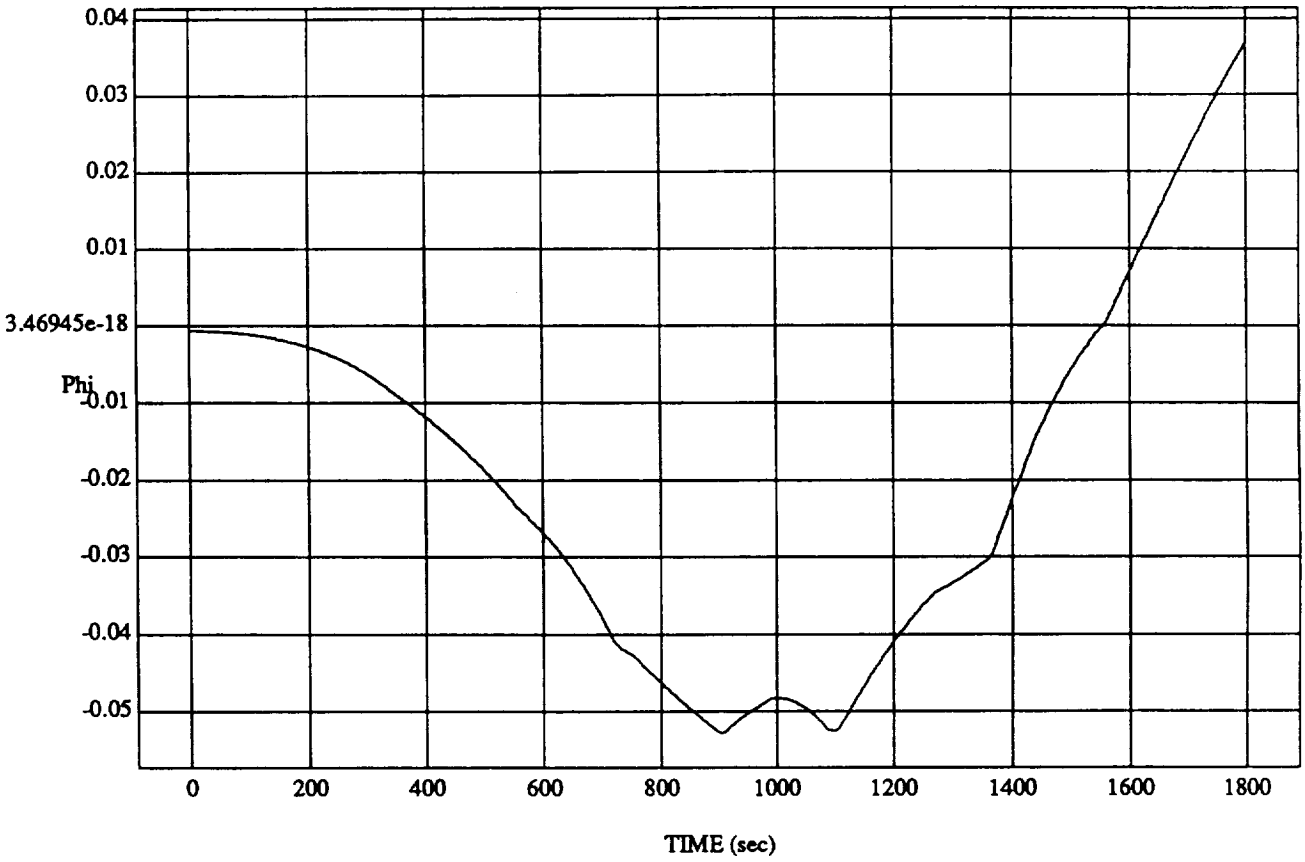


MODULE: ORBITER.lm_elev
DATA SAMPLING FREQUENCY: 0.200 Hz

SIMULATION APPLICATION: ARIC Translational Controller Simulation

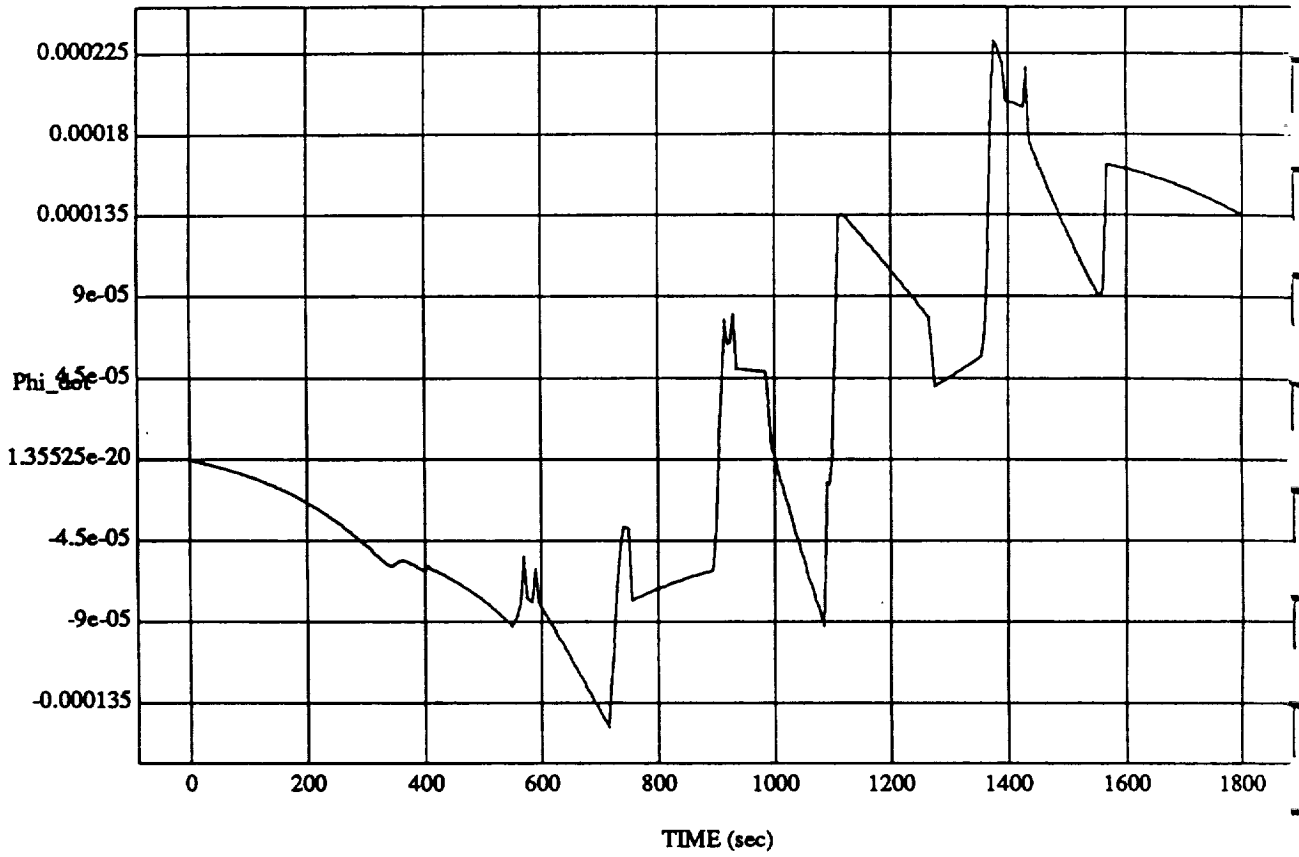
Phi vs TIME

RUN: V Bar Approach

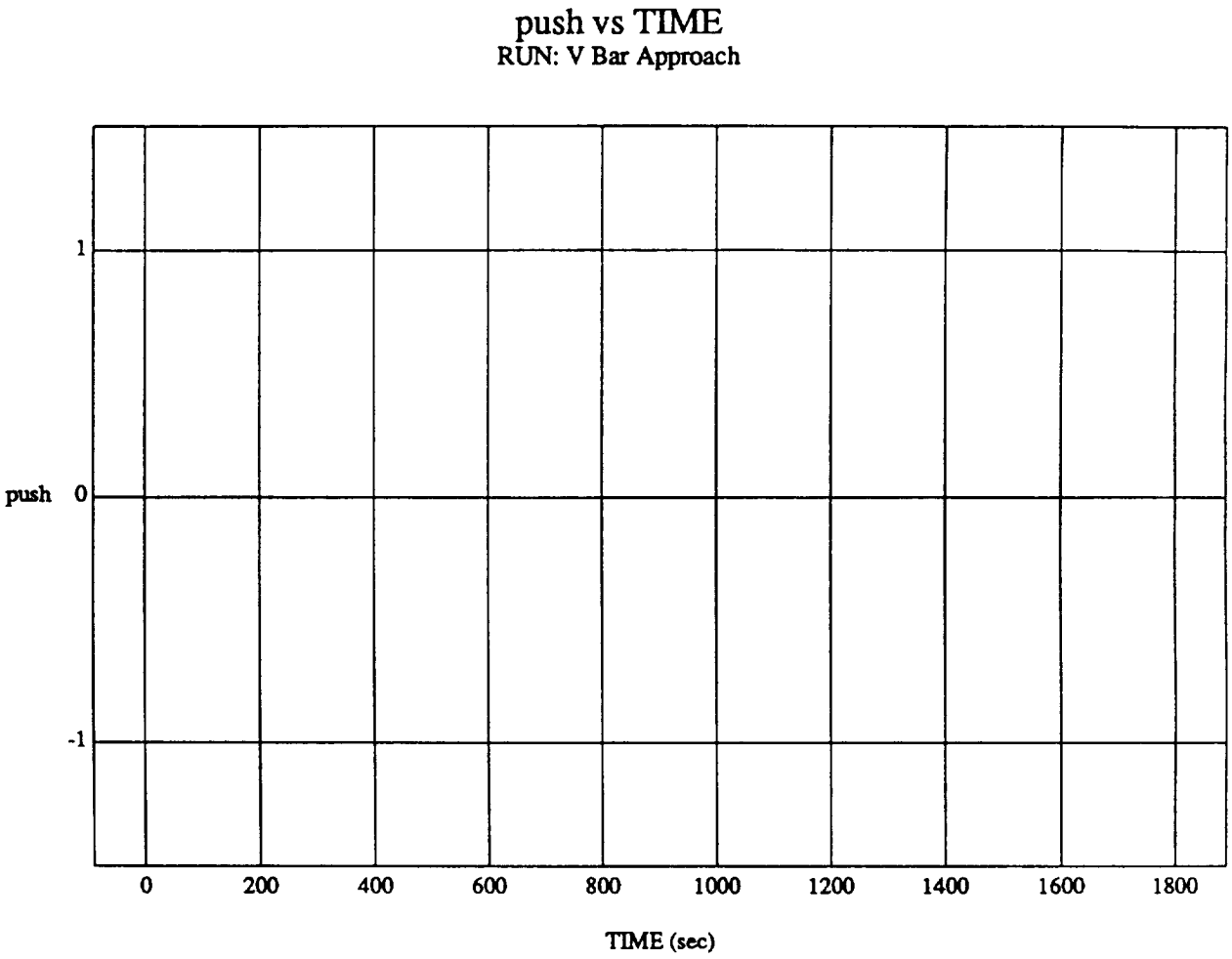


MODULE: ORBITER.lm_azim
DATA SAMPLING FREQUENCY: 0.200 Hz

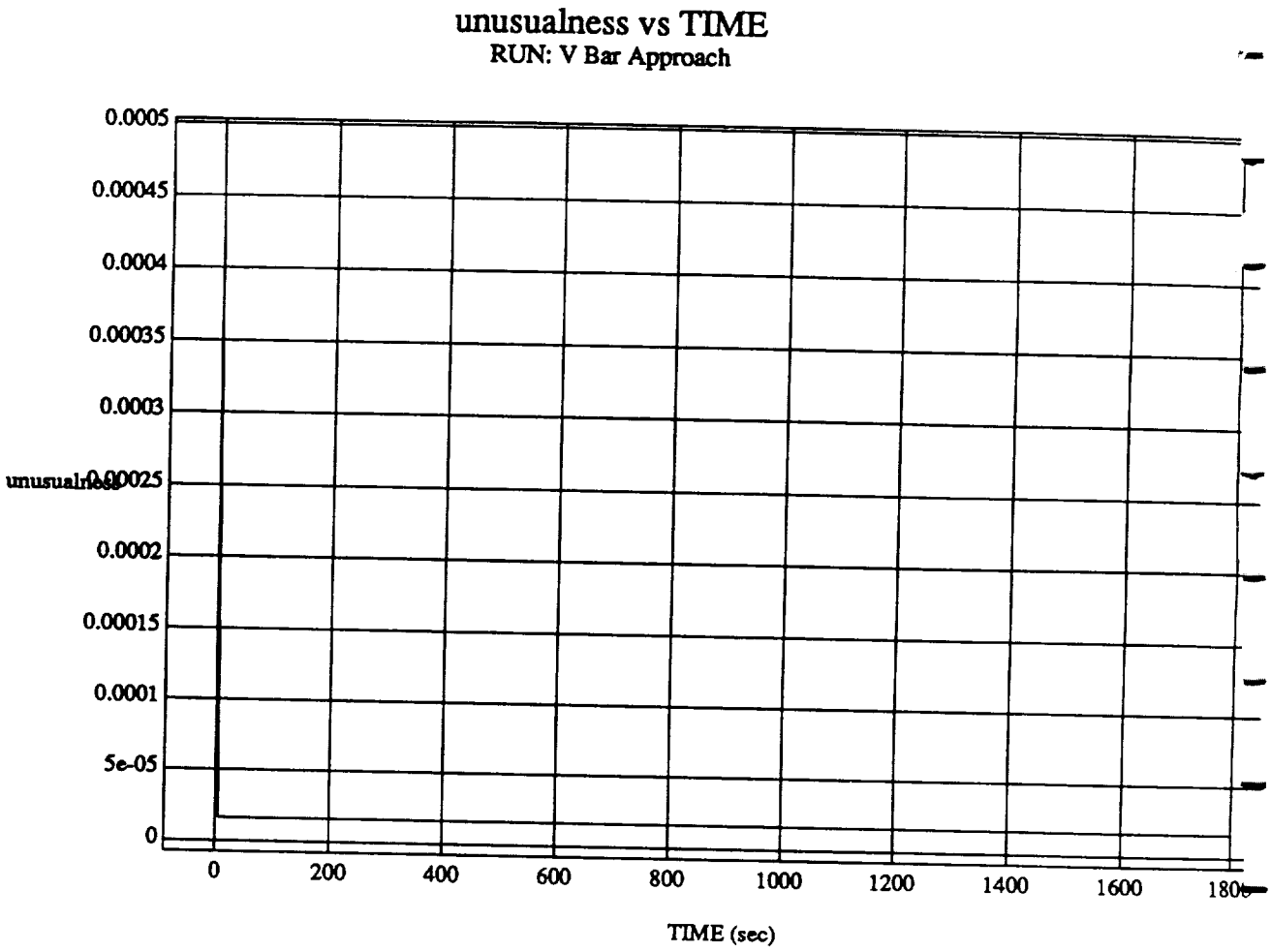
Phi_dot vs TIME
RUN: V Bar Approach



MODULE: ORBITER.lm_azim
DATA SAMPLING FREQUENCY: 0.200 Hz



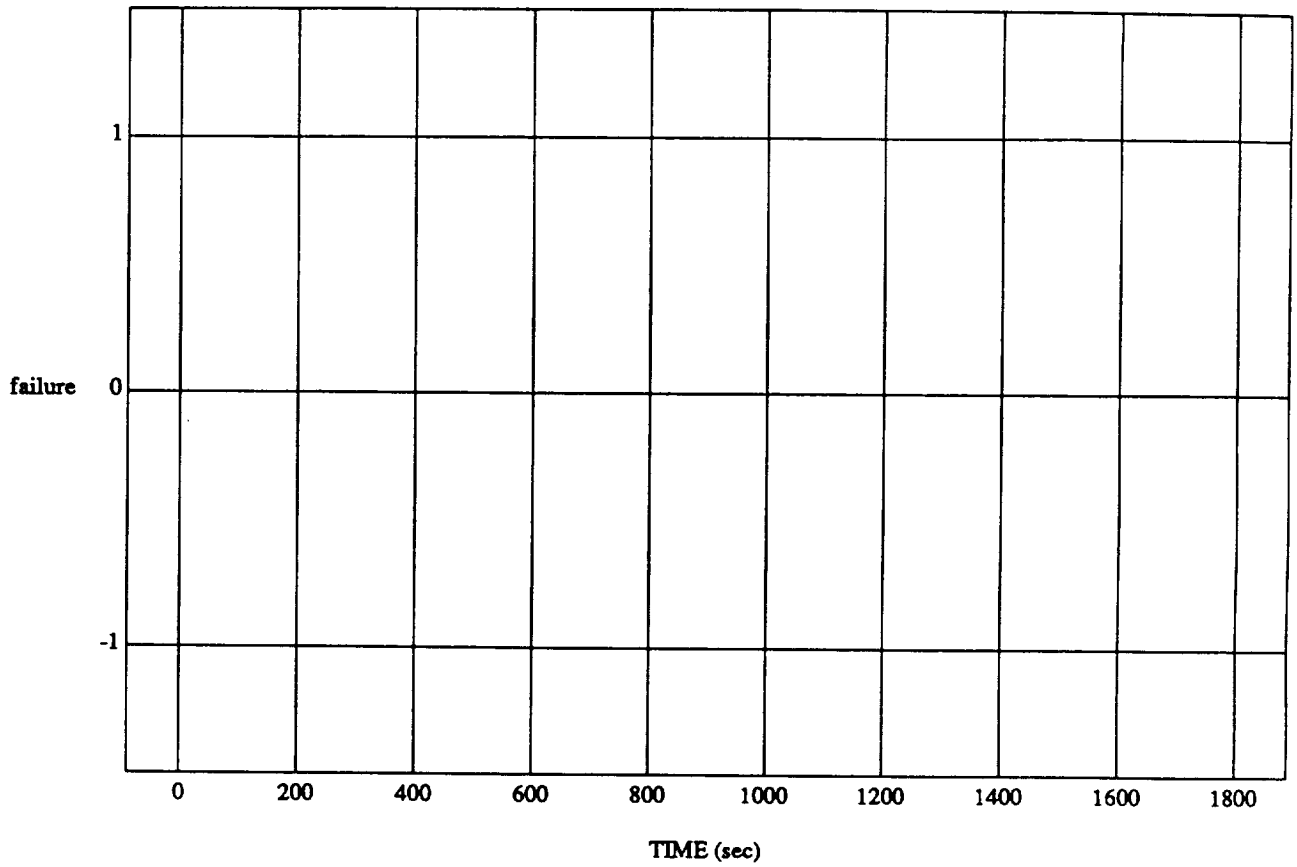
MODULE: ORBITER.lm_azim
DATA SAMPLING FREQUENCY: 0.200 Hz



MODULE: ORBITER.lm_azim
DATA SAMPLING FREQUENCY: 0.200 Hz

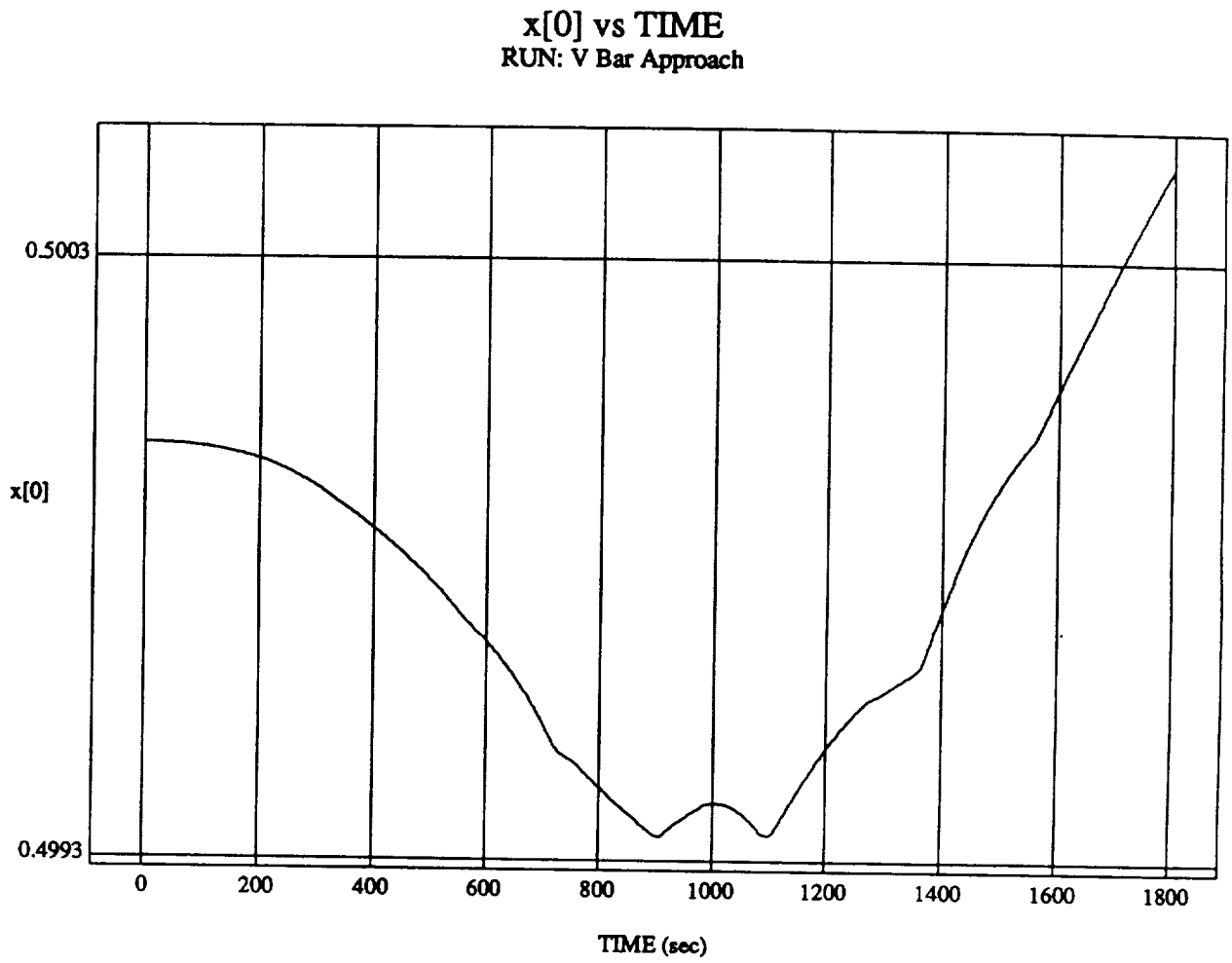
failure vs TIME

RUN: V Bar Approach

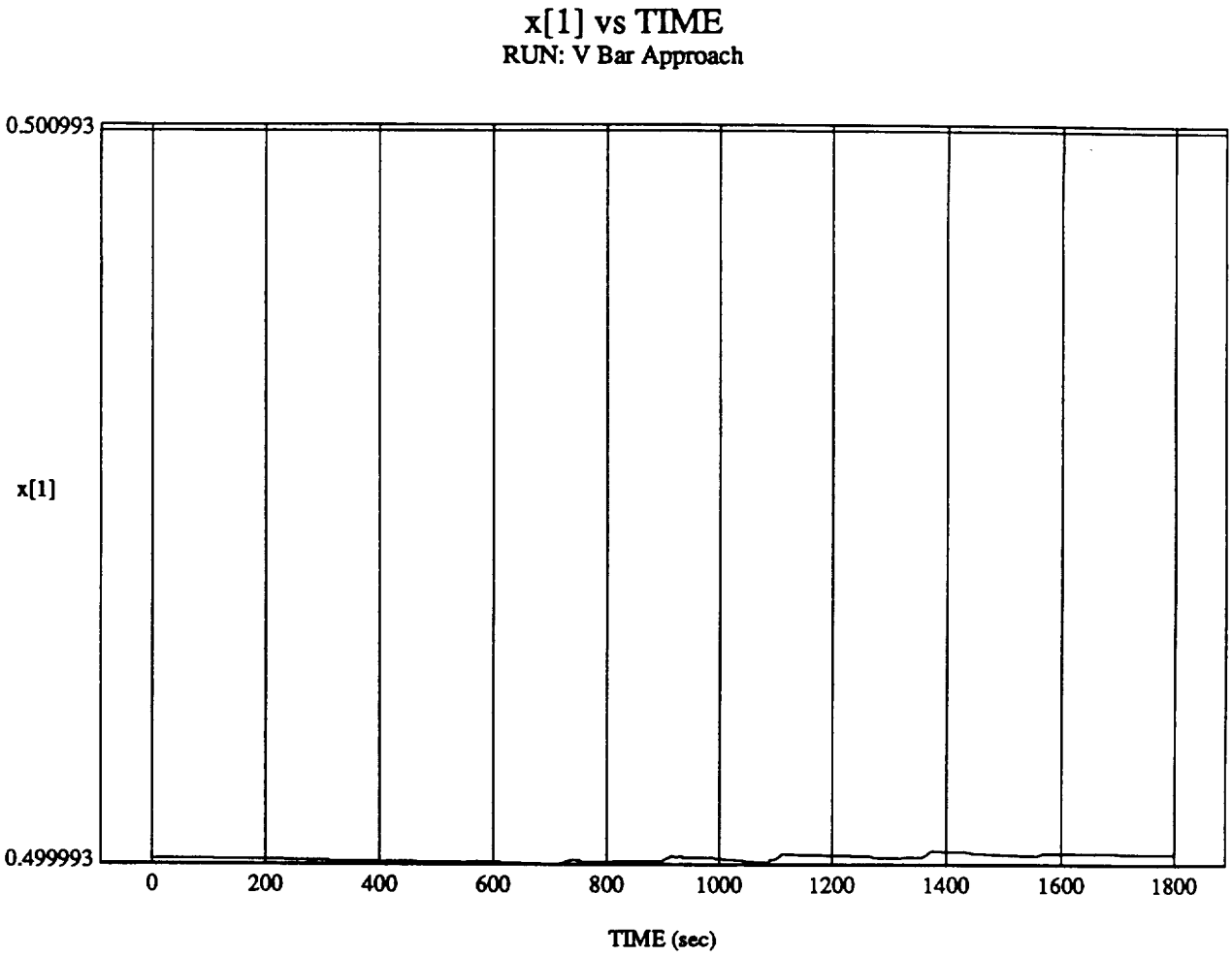


MODULE: ORBITER.lm_azim
DATA SAMPLING FREQUENCY: 0.200 Hz

SIMULATION APPLICATION: ARIC Translational Controller Simulation



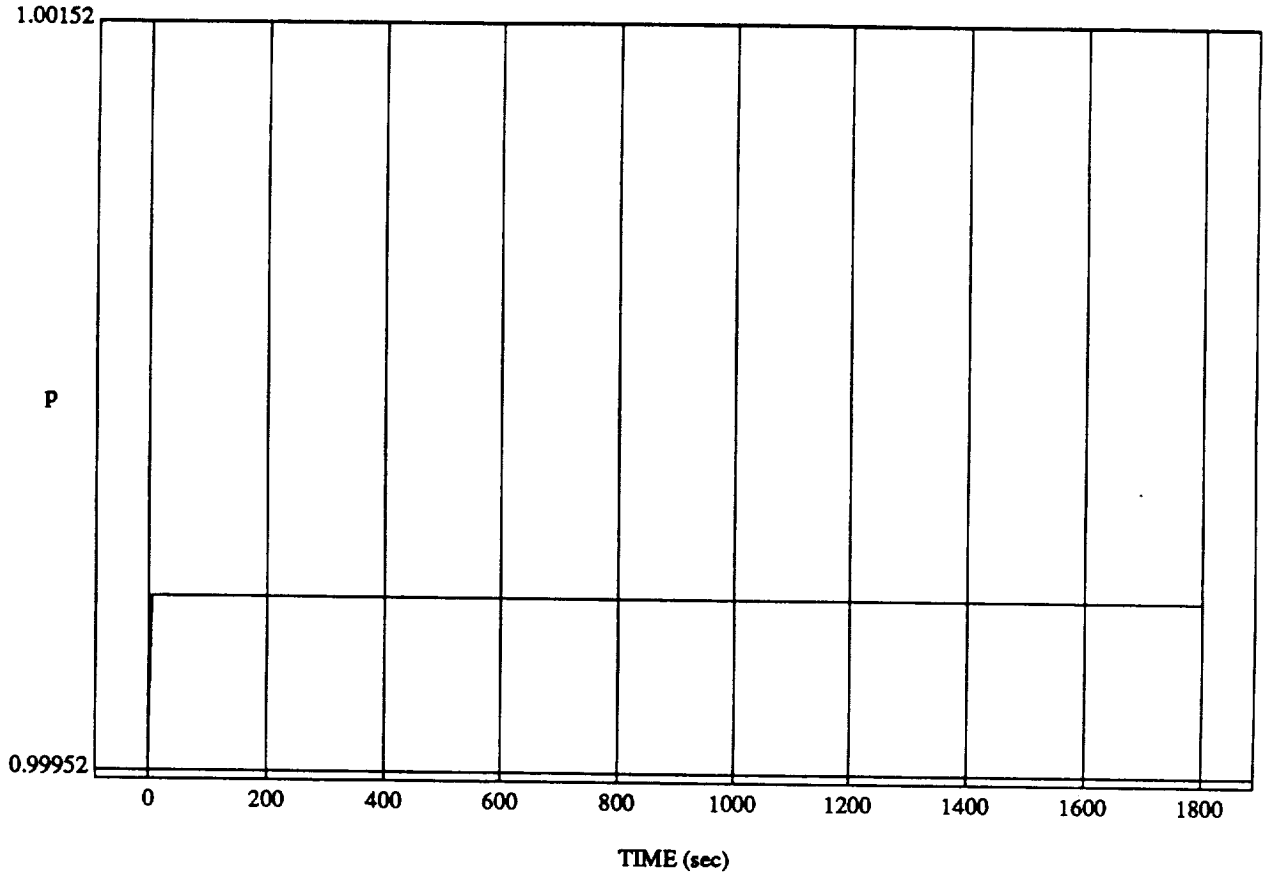
MODULE: ORBITER.lm_azim
DATA SAMPLING FREQUENCY: 0.200 Hz



MODULE: ORBITER.lm_azim
DATA SAMPLING FREQUENCY: 0.200 Hz

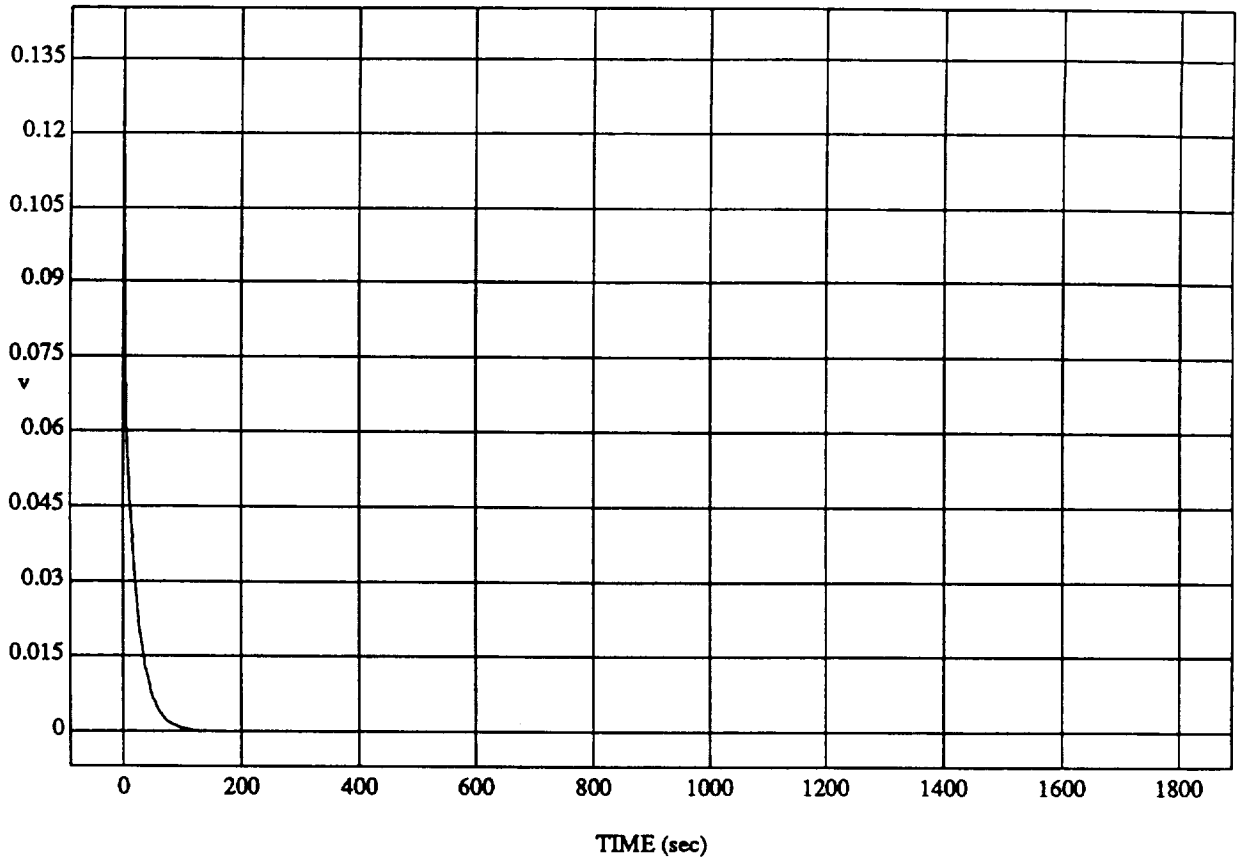
SIMULATION APPLICATION: ARIC Translational Controller Simulation

p vs TIME
RUN: V Bar Approach



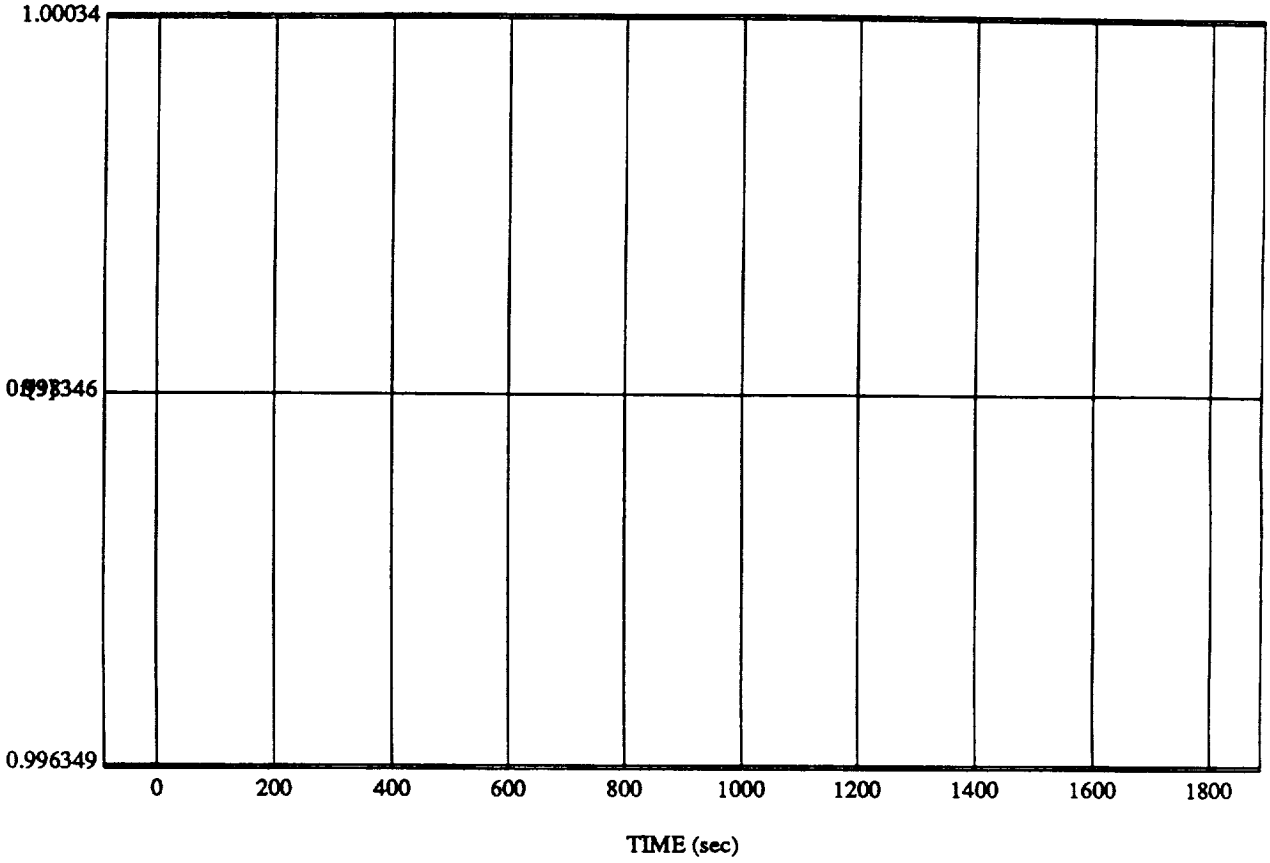
MODULE: ORBITER.lm_azim
DATA SAMPLING FREQUENCY: 0.200 Hz

v vs TIME
RUN: V Bar Approach



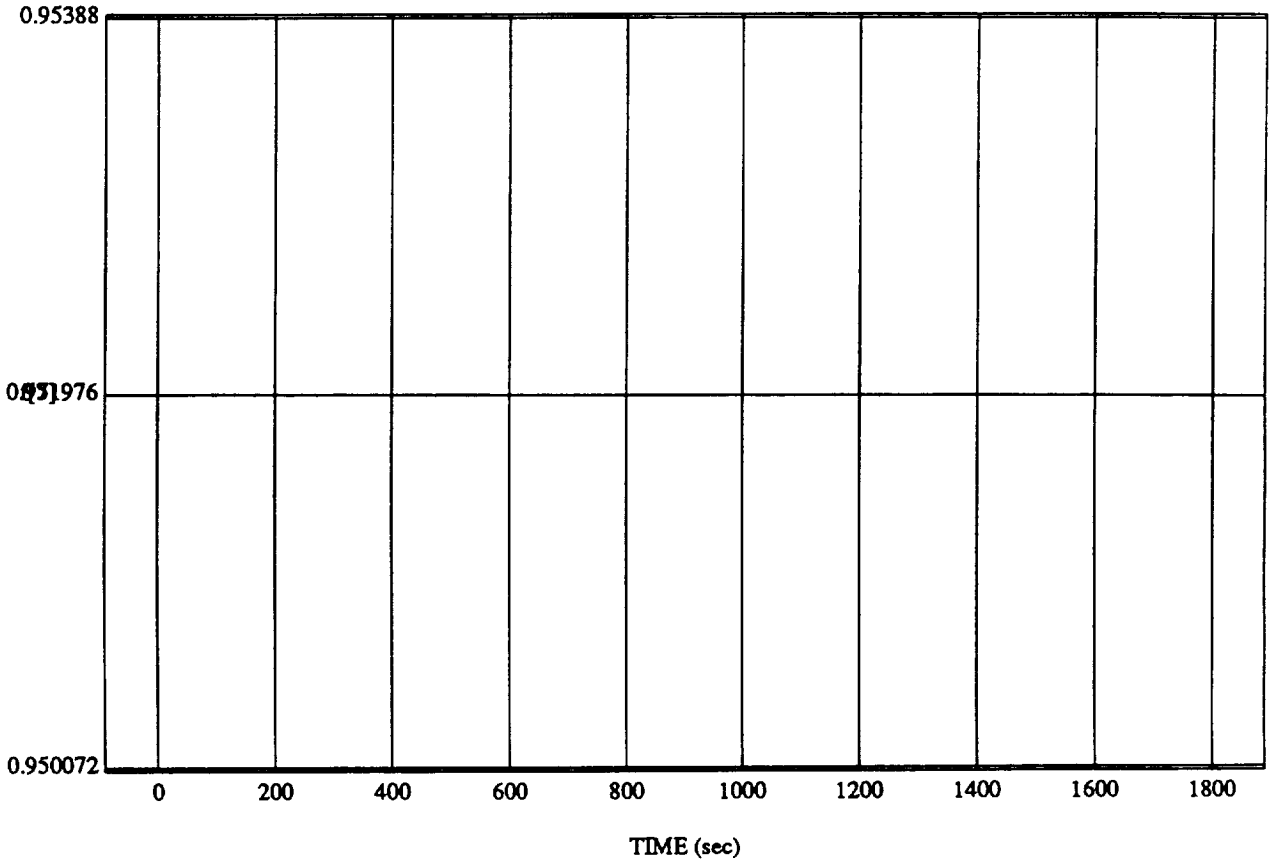
MODULE: ORBITER.lm_azim
DATA SAMPLING FREQUENCY: 0.200 Hz

f[3] vs TIME
RUN: V Bar Approach

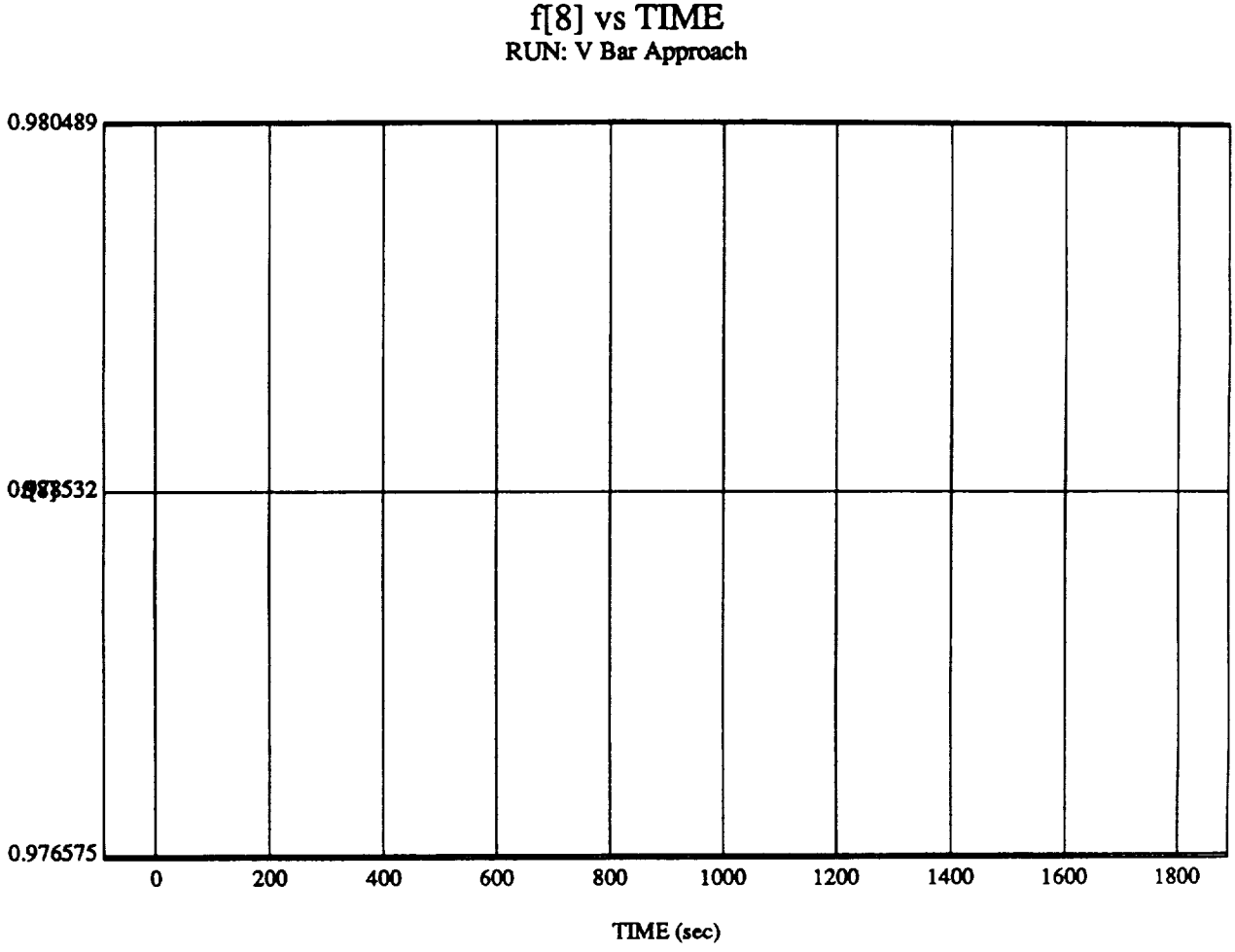


MODULE: ORBITER.lm_azim
DATA SAMPLING FREQUENCY: 0.200 Hz

f[7] vs TIME
RUN: V Bar Approach

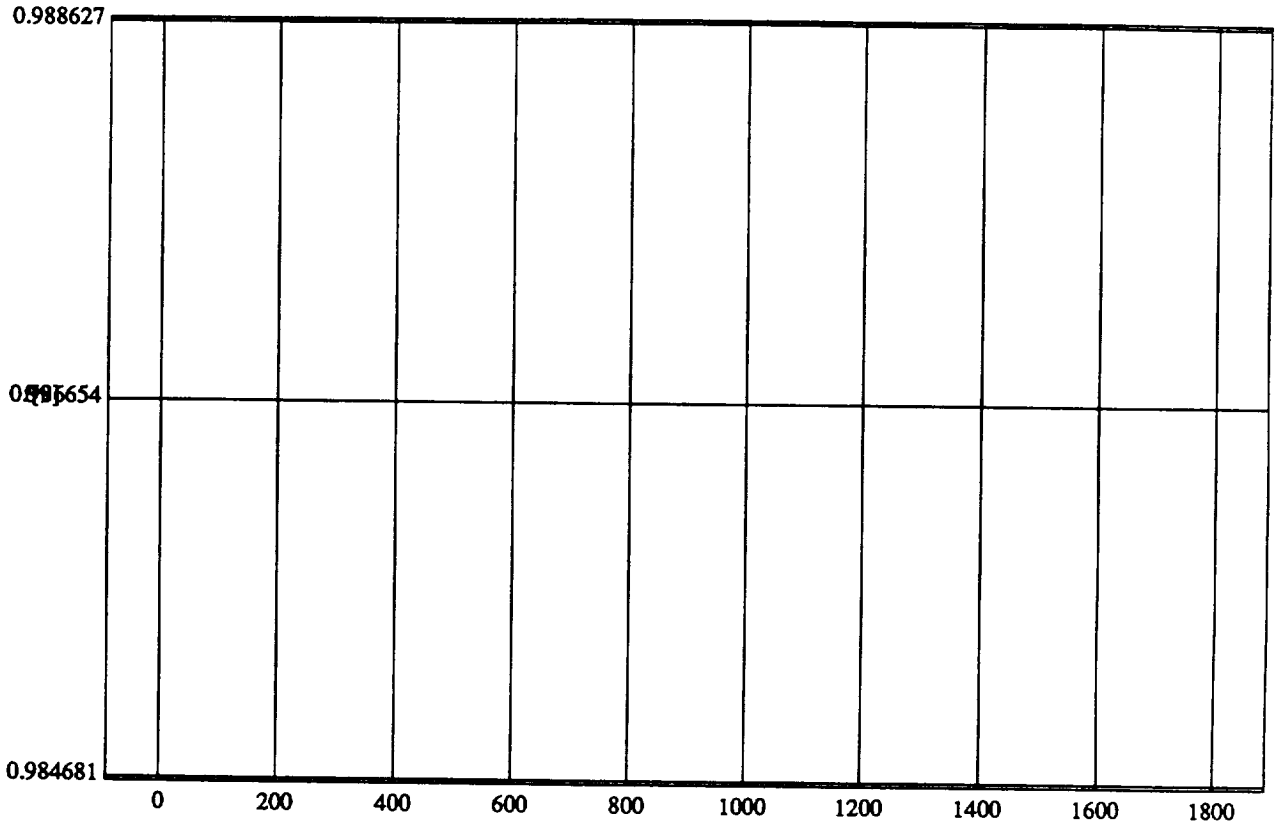


MODULE: ORBITER.lm_azim
DATA SAMPLING FREQUENCY: 0.200 Hz



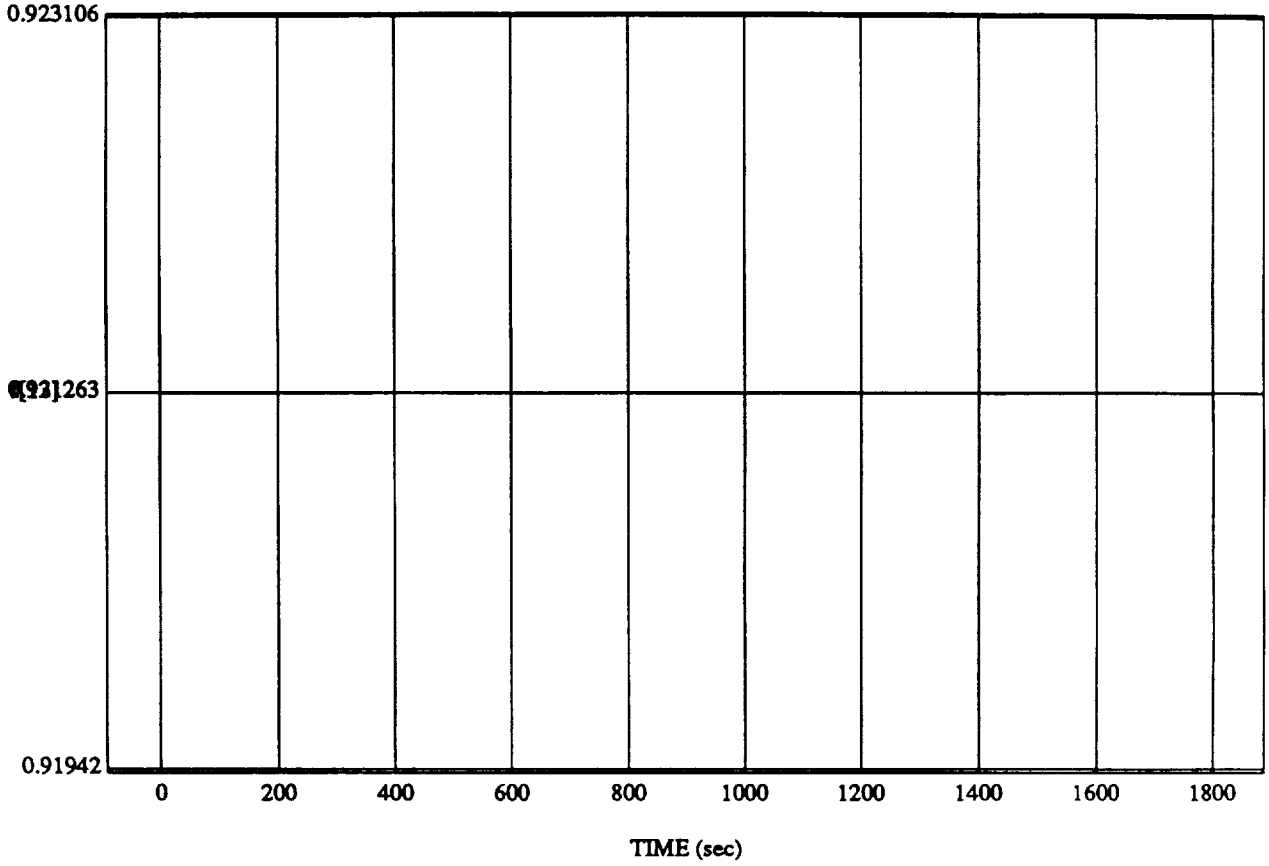
MODULE: ORBITER.lm_azim
DATA SAMPLING FREQUENCY: 0.200 Hz

f[9] vs TIME
RUN: V Bar Approach



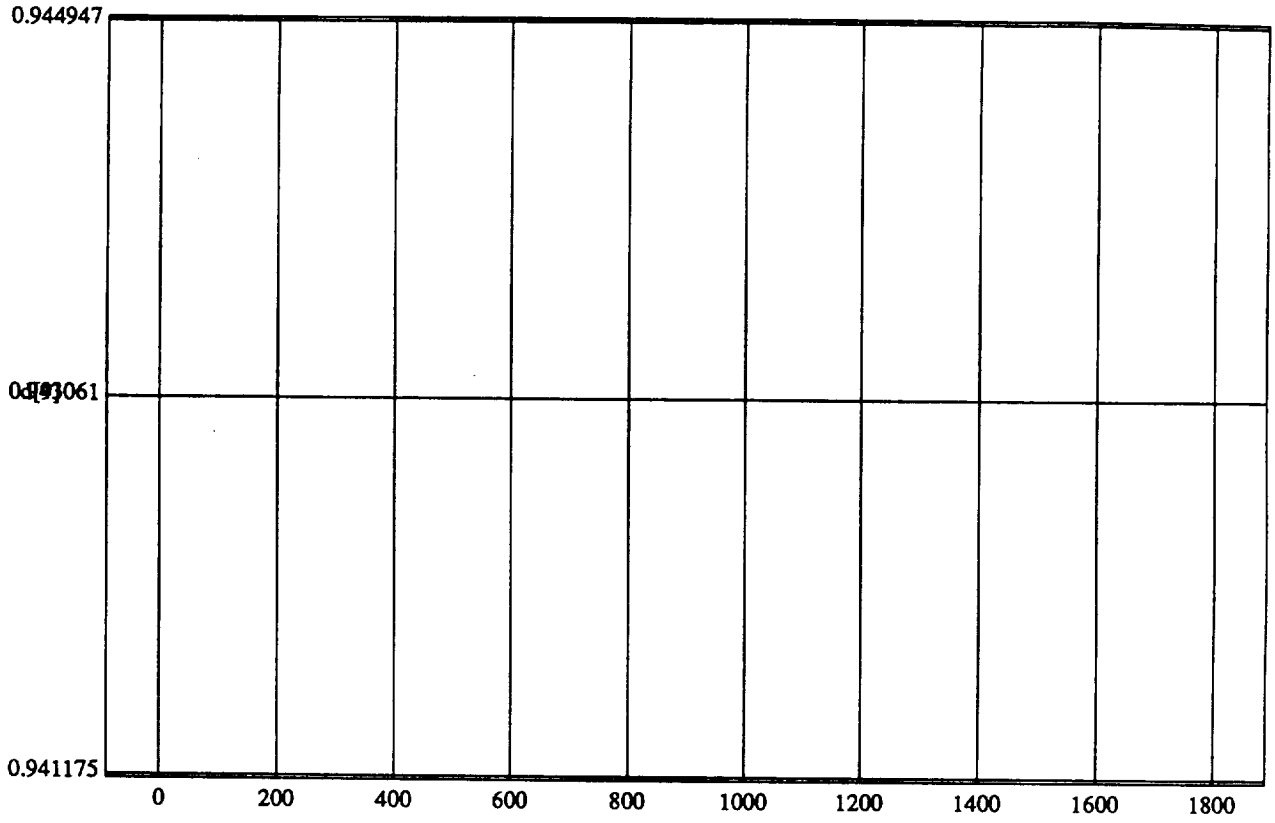
MODULE: ORBITER.lm_azim
DATA SAMPLING FREQUENCY: 0.200 Hz

f[13] vs TIME
RUN: V Bar Approach



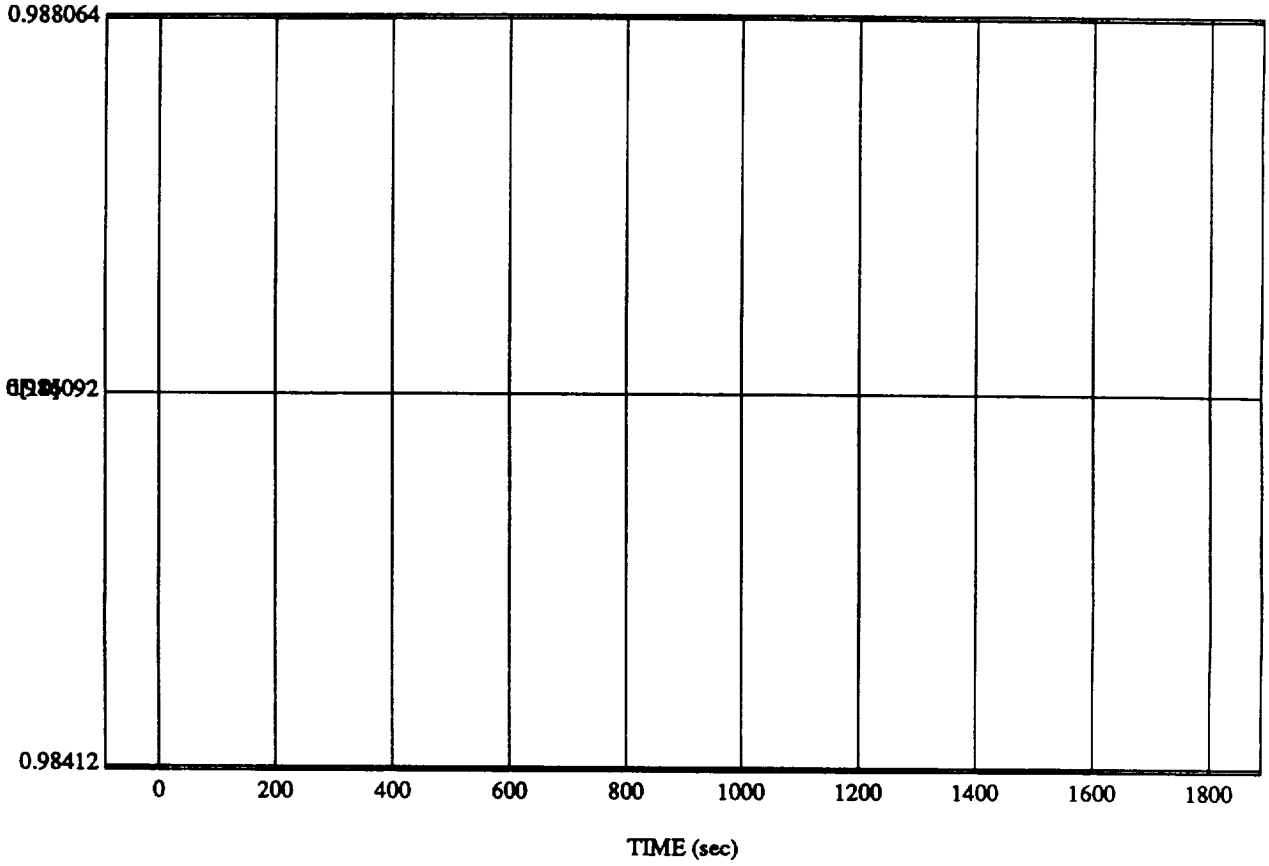
MODULE: ORBITER.lm_azim
DATA SAMPLING FREQUENCY: 0.200 Hz

d[9] vs TIME
RUN: V Bar Approach

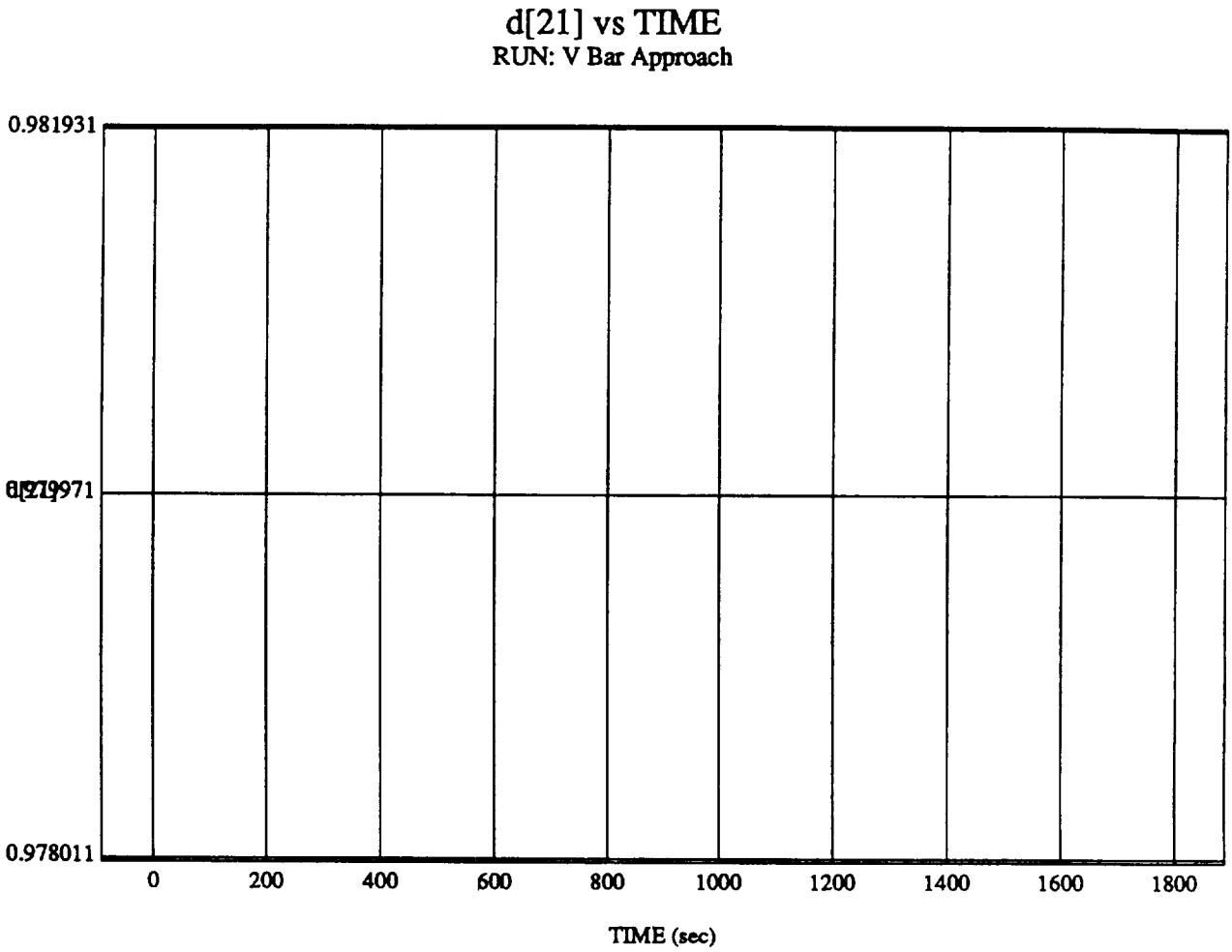


MODULE: ORBITER.lm_azim
DATA SAMPLING FREQUENCY: 0.200 Hz

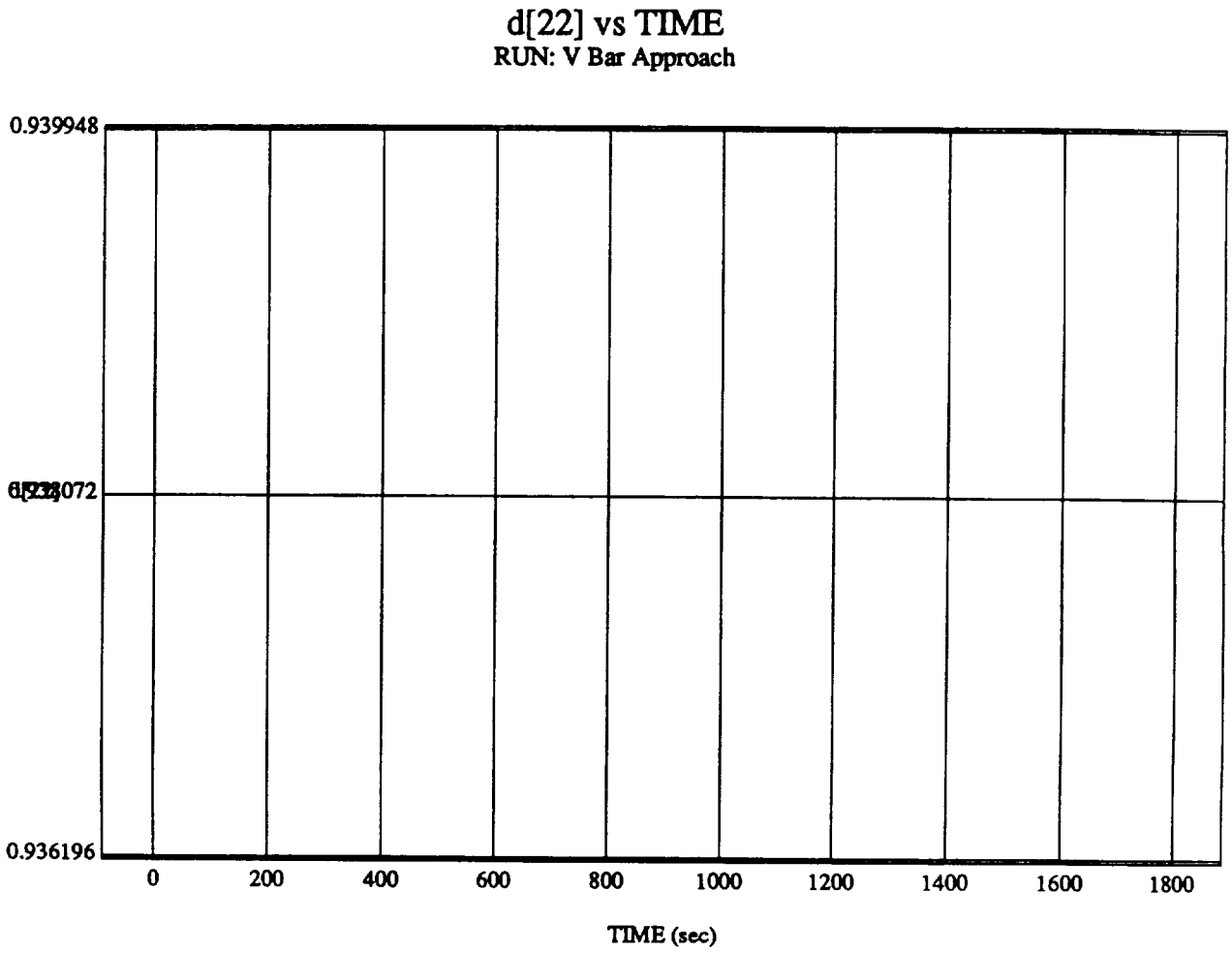
d[10] vs TIME
RUN: V Bar Approach



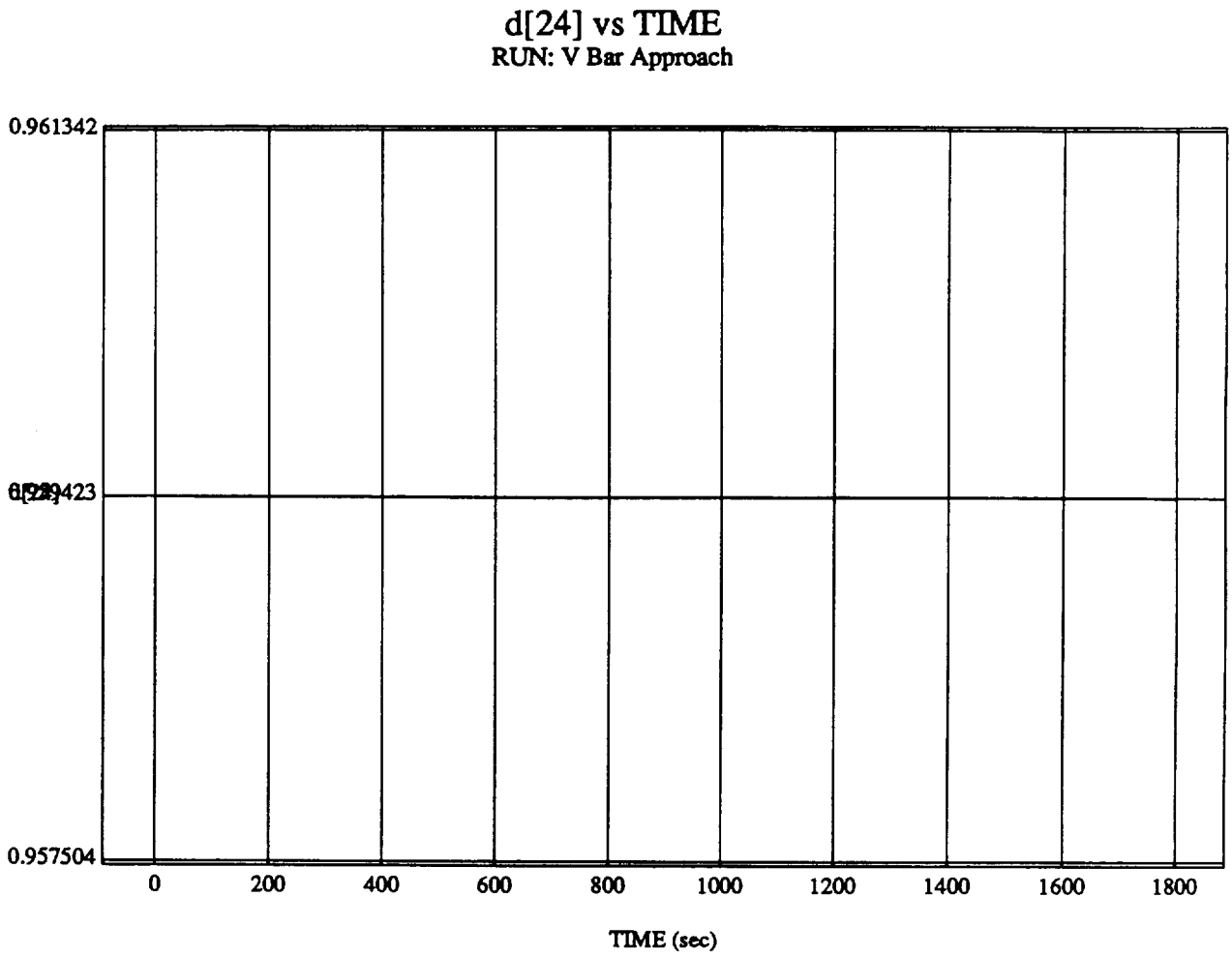
MODULE: ORBITER.lm_azim
DATA SAMPLING FREQUENCY: 0.200 Hz



MODULE: ORBITER.lm_azim
DATA SAMPLING FREQUENCY: 0.200 Hz

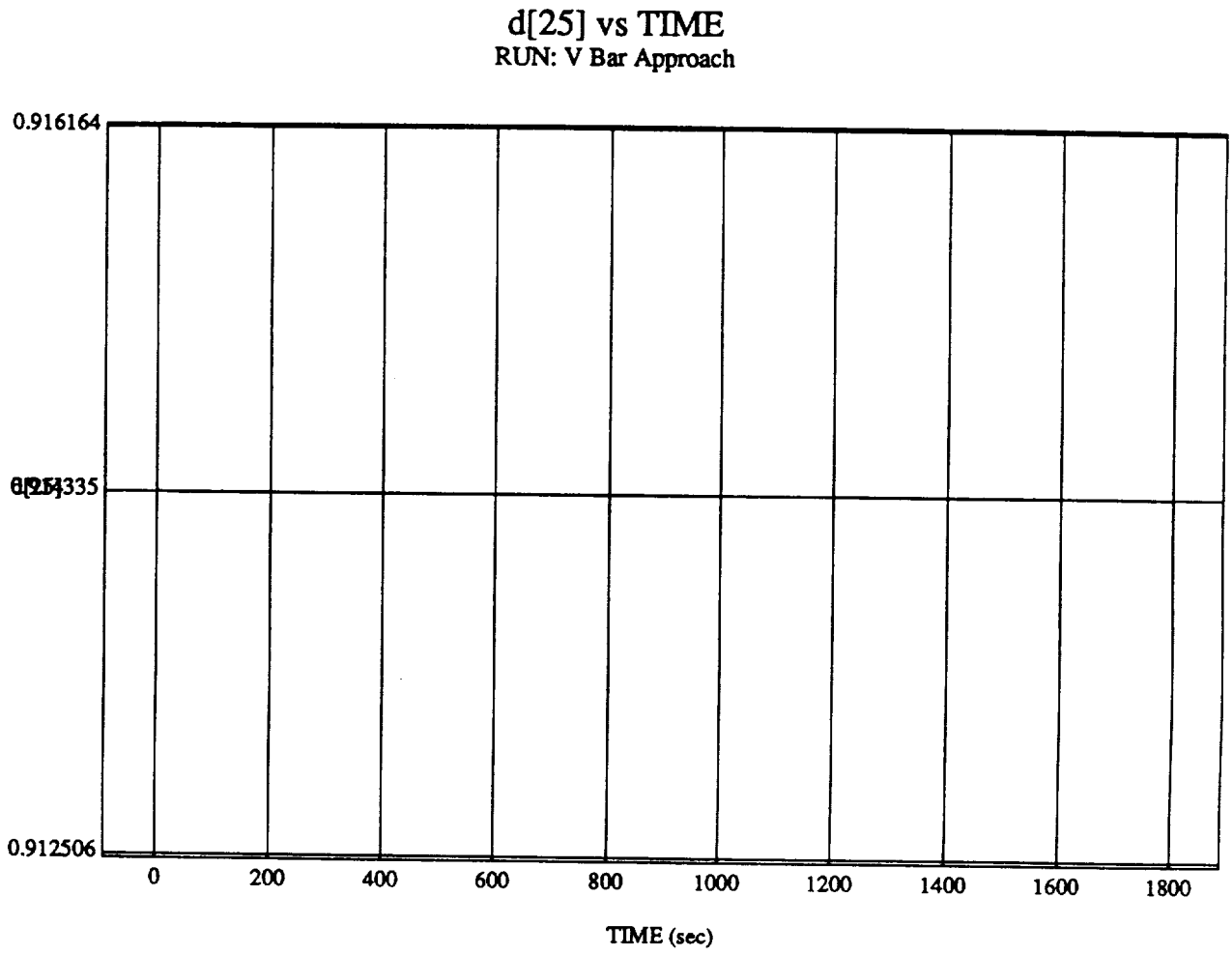


MODULE: ORBITER.lm_azim
DATA SAMPLING FREQUENCY: 0.200 Hz

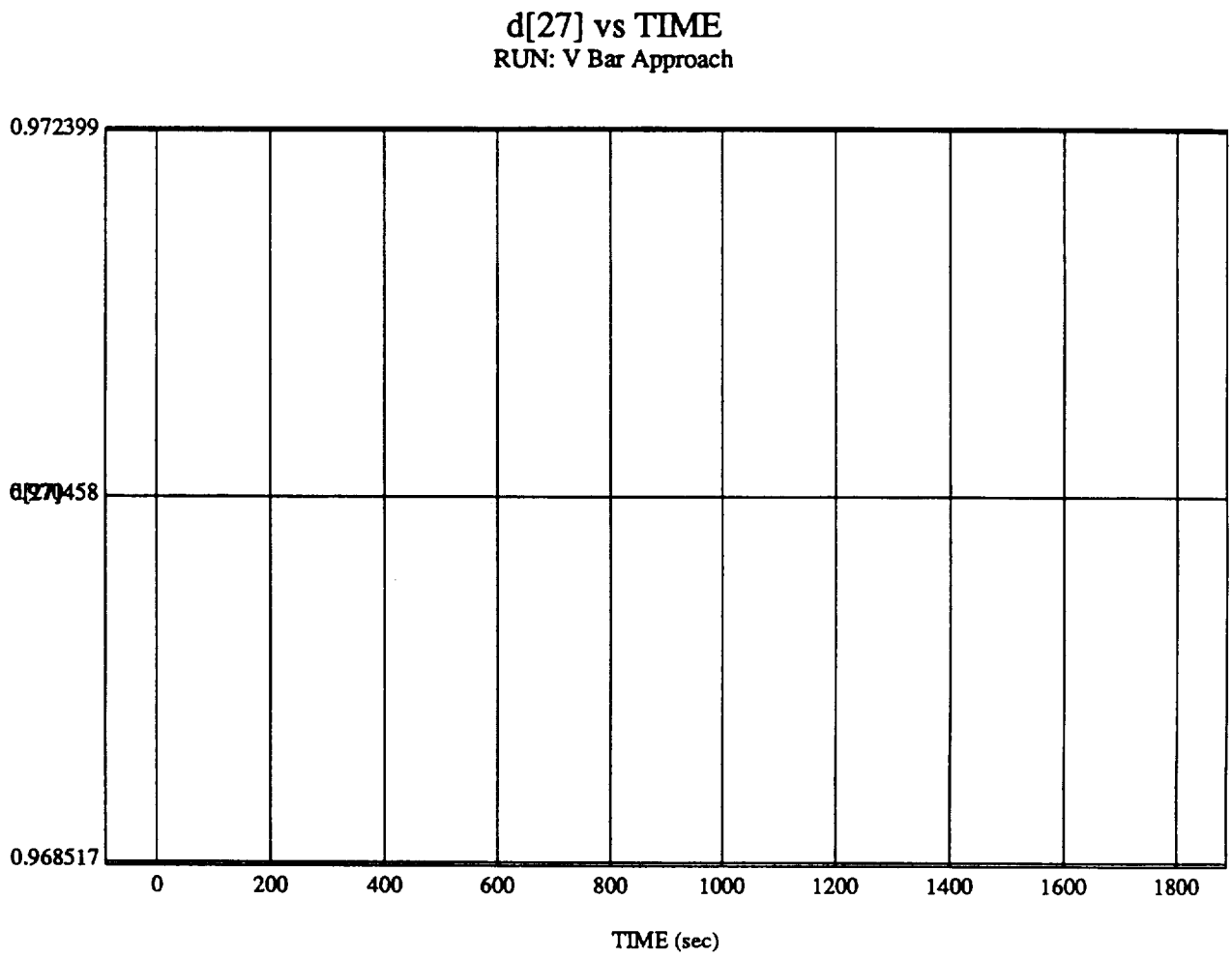


MODULE: ORBITER.lm_azim
DATA SAMPLING FREQUENCY: 0.200 Hz

SIMULATION APPLICATION: ARIC Translational Controller Simulation

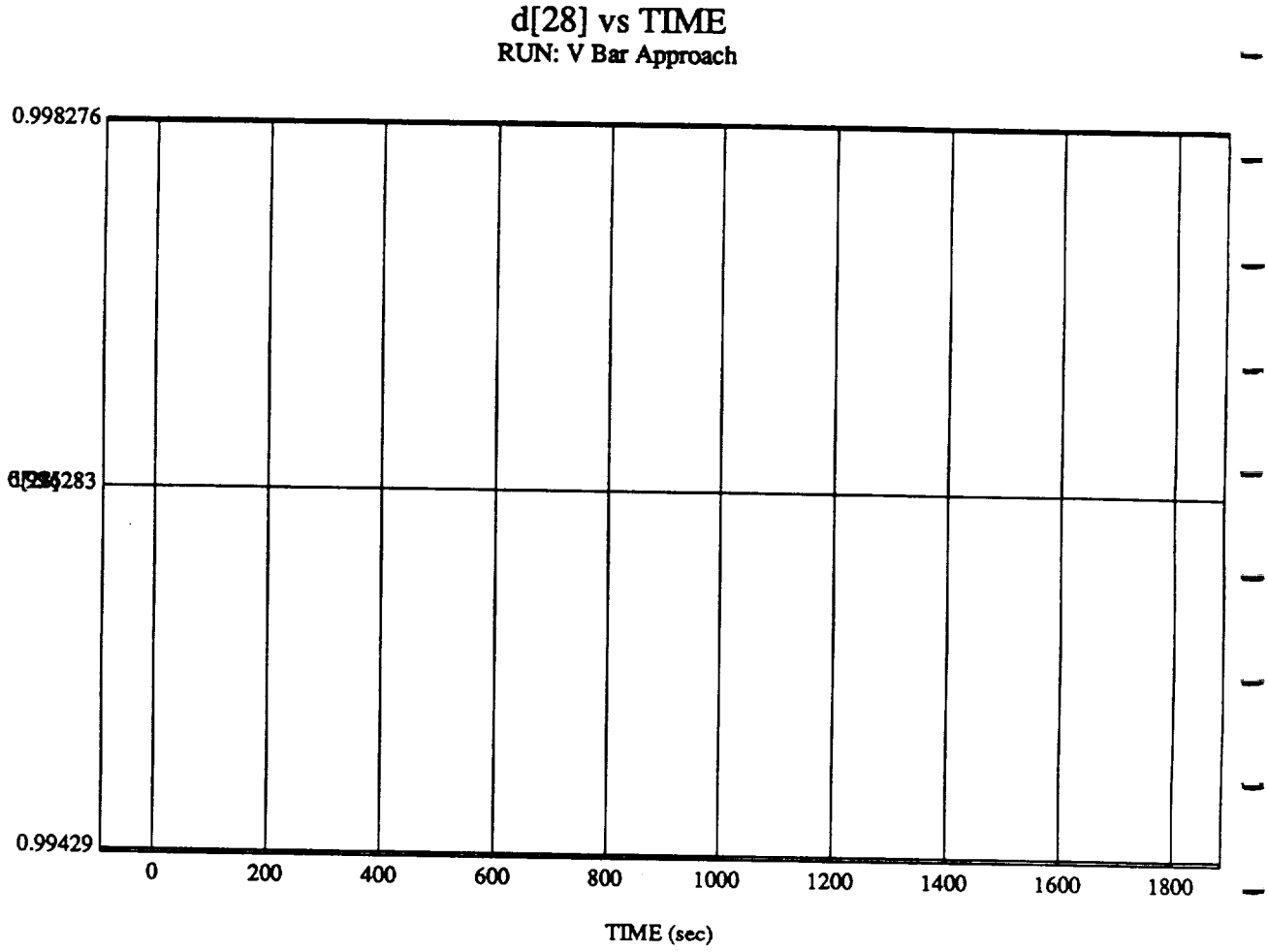


MODULE: ORBITER.lm_azim
DATA SAMPLING FREQUENCY: 0.200 Hz

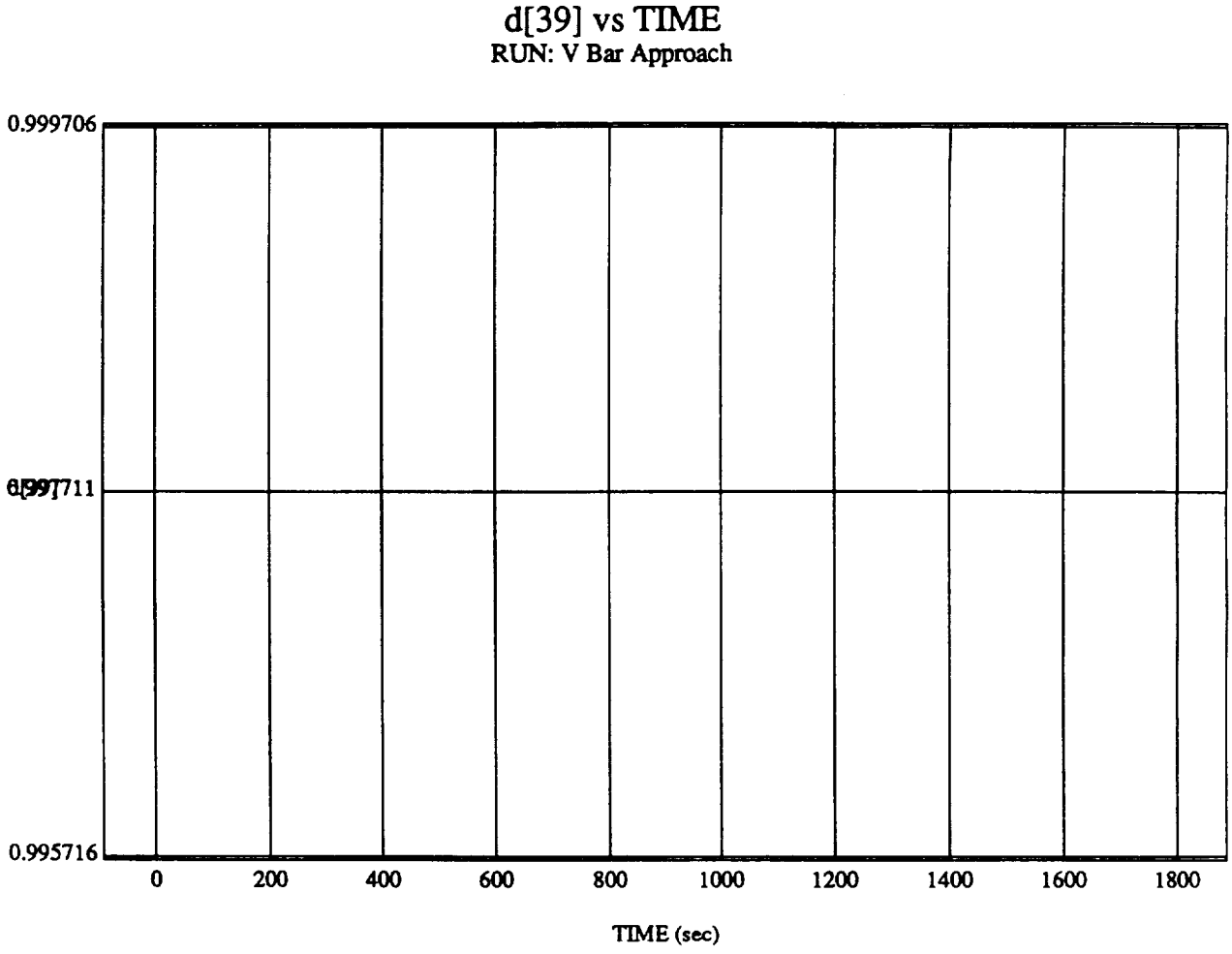


MODULE: ORBITER.lm_azim
DATA SAMPLING FREQUENCY: 0.200 Hz

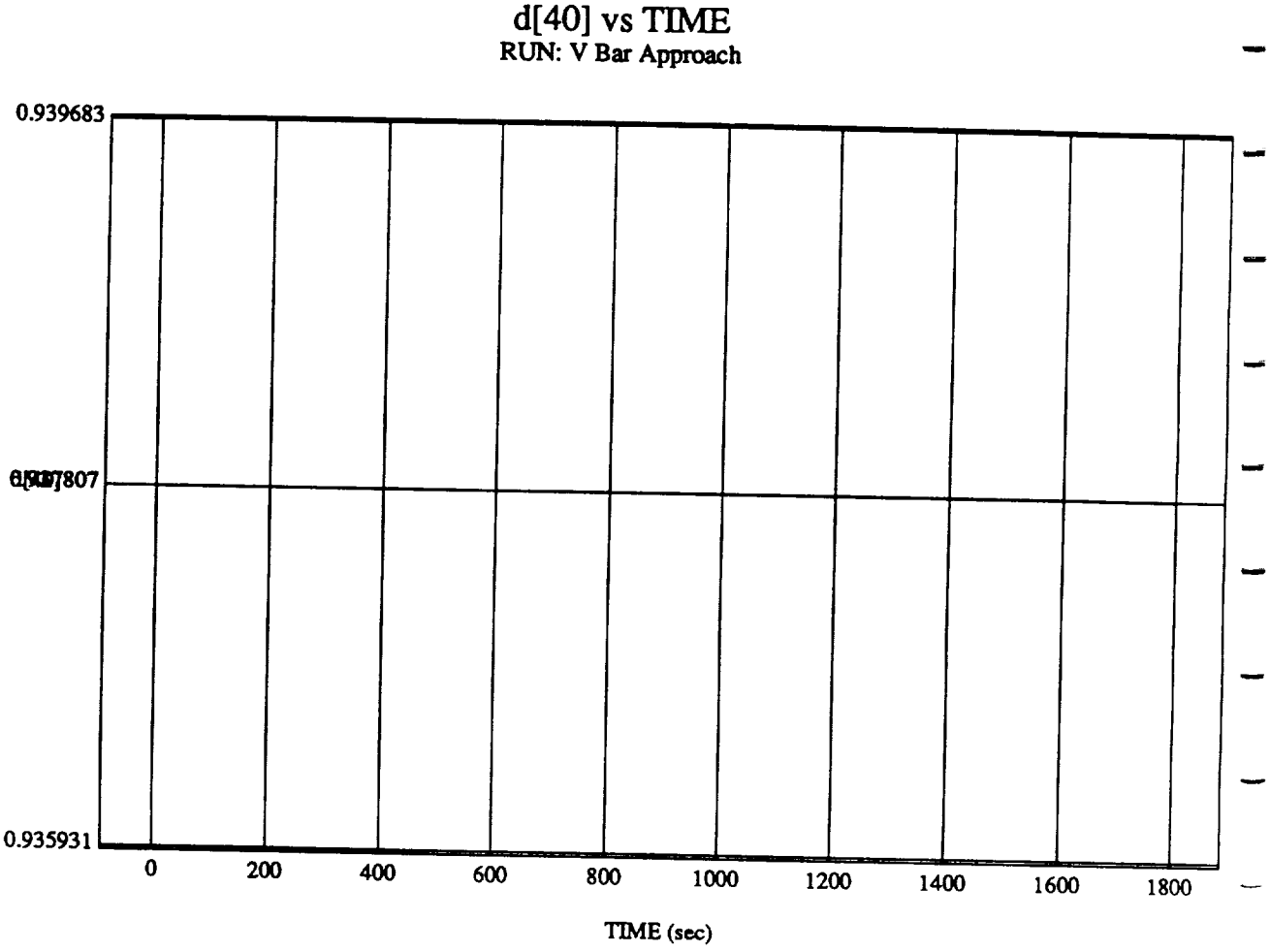
SIMULATION APPLICATION: ARIC Translational Controller Simulation



MODULE: ORBITER.lm_azim
DATA SAMPLING FREQUENCY: 0.200 Hz



MODULE: ORBITER.lm_azim
DATA SAMPLING FREQUENCY: 0.200 Hz



MODULE: ORBITER.lm_azim
DATA SAMPLING FREQUENCY: 0.200 Hz